DESIGN AND ANALYSIS OF ALGORITHM [23CS501]

UNIT I:

Introduction: Algorithm, Psuedo code for expressing algorithms, Performance Analysis- Space complexity, Time complexity, Asymptotic Notation- Big oh notation, Omega notation,Theta notation and Little oh notation.

Divide and conquer: General method, applications-Binary search, Quick sort, Merge sort,Strassen's matrix multiplication.

INTRODUCTION TO ALGORITHM

History of Algorithm

The word algorithm comes from the name of a Persian author, Abu Ja'far Mohammed ibnMusa al Khowarizmi (c. 825 A.D.), who wrote a textbook on mathematics.
He is credited with providing the step-by-step rules for adding, subtracting, multiplying,and dividing ordinary decimal numbers.
When written in Latin, the name became Algorismus, from which algorithm is but a smallstep
This word has taken on a special significance in computer science, where "algorithm" has come to refer to a method that can be used by a computer for the solution of a problem
Between 400 and 300 B.C., the great Greek mathematician Euclid invented an algorithm
Finding the greatest common divisor (gcd) of two positive integers.
The gcd of X and Y is the largest integer that exactly divides both X and Y .
Eg.,the gcd of 80 and 32 is 16.
The Euclidian algorithm, as it is called, is considered to be the first non-trivial algorithmever devised.

What is an Algorithm?

Algorithm is a set of steps to complete a task.

For example,

Task: to make a cup of tea.
Algorithm:
add water and milk to the kettle,
boil it, add tea leaves,
Add sugar, and then serve it in cup.

''a set of steps to accomplish or complete a task that is described precisely enough thata computer can run it''.

Described precisely: very difficult for a machine to know how much water, milk to be added etc. in the above tea making algorithm.

These algorithms run on computers or computational devices For example, GPS in our smartphones, Google hangouts.

GPS uses shortest path algorithm.. **Online shopping** uses cryptography which uses RSA algorithm.

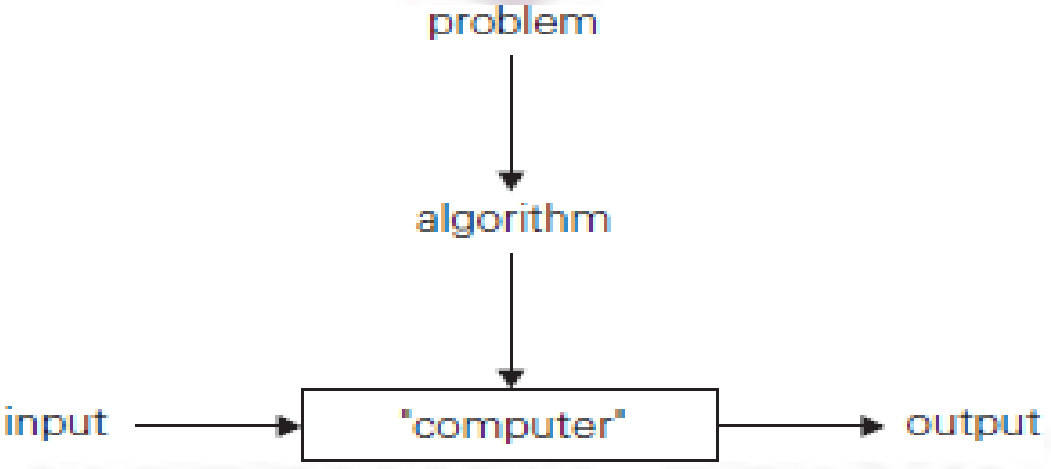
Algorithm Definition1:

An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:
Input. Zero or more quantities are externally supplied.
Output. At least one quantity is produced.
Definiteness. Each instruction is clear and unambiguous.
Finiteness. The algorithm terminates after a finite number of steps.
Effectiveness. Every instruction must be very basic enough and must befeasible.

Algorithm Definition2:

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., forobtaining a required output for any legitimate input in a finite amount of time.

Algorithms that are definite and effective are also called *computational procedures*.
A program is the expression of an algorithm in a programming language



Algorithms for Problem Solving

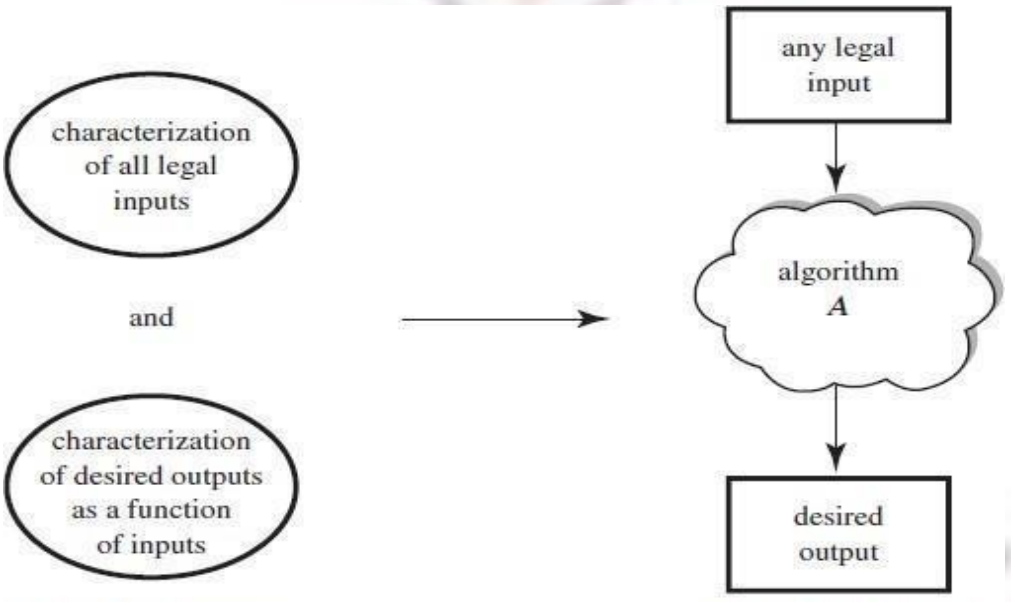
The main steps for Problem Solving are:
Problem definition
Algorithm design / Algorithm specification
Algorithm analysis
Implementation
Testing
[Maintenance]
Step1. Problem Definition
What is the task to be accomplished?
Ex: Calculate the average of the grades for a given student

Step2.Algorithm Design / Specifications:

Describe: in natural language / pseudo-code / diagrams / etc

Step3. Algorithm analysis
Space complexity - How much space is required
Time complexity - How much time does it take to run the algorithmComputer Algorithm
An algorithm is a procedure (a finite set of well-defined instructions) for accomplishingsome tasks which, given an initial state terminate in a defined end-state
The computational complexity and efficient implementation of the algorithm are importantin computing, and this depends on suitable data structures.
Steps 4,5,6: Implementation, Testing, Maintainance
Implementation:
Decide on the programming language to use C, C++, Lisp, Java, Perl, Prolog, assembly, etc.
, etc.
Write clean, well documented code
Test, test, test
Integrate feedback from users, fix bugs, ensure compatibility across different versions
Maintenance.Release Updates,fix bugs

Keeping illegal inputs separate is the responsibility of the algorithmic problem, while treating special classes of unusual or undesirable inputs is the responsibility of the algorithm itself.



4 Distinct areas of study of algorithms:

How to devise algorithms. └ Techniques – Divide & Conquer, Branch and Bound ,Dynamic Programming
How to validate algorithms.
Check for Algorithm that it computes the correct answer for all possible legal inputs. └ algorithm validation. □ First Phase
Second phase └ Algorithm to Program └ Program Proving or Program Verification
□ Solution be stated in two forms:
First Form: Program which is annotated by a set of assertions about the input and outputvariables of the program └ predicate calculus

Second form: is called a specification
4 Distinct areas of study of algorithms (..Contd)
How to analyze algorithms.
Analysis of Algorithms or performance analysis refer to the task of determining howmuch computing time & storage an algorithm requires
How to test a program ⌊ 2 phases ⌊
Debugging - Debugging is the process of executing programs on sample data sets todetermine whether faulty results occur and, if so, to correct them.
Profiling or performance measurement is the process of executing a correct program ondata sets and measuring the time and space it takes to compute the results

PSEUDOCODE:

Algorithm can be represented in Text mode and Graphic mode
Graphical representation is called Flowchart
Text mode most often represented in close to any High level language such as C, Pascal ⌊ Pseudocode

Pseudocode: High-level description of an algorithm.

⌋More structured than plain English.
⌊ Less detailed than a program.
⌊ Preferred notation for describing algorithms.
⌊ Hides program design issues.

Example of Pseudocode:

To find the max element of an array
Algorithm *arrayMax*(*A*, *n*) Input array *A* of *n* integers Output maximum element of *A*
currentMax ⌊ *A*[0]
for *i* ⌋ 1 to *n* ⌋ 1 do
if *A*[*i*] ⌊ *currentMax* then
currentMax ⌊ *A*[*i*]

return *currentMax*
Control flow
if … then … [else …]
while … do …
repeat … until …
for … do …
Indentation replaces braces
Method declaration
Algorithm *method* (*arg* [, *arg*…])
Input …
Output …
Method call
var.method (*arg* [, *arg*…])
Return value
return *expression*
Expressions
Assignment (equivalent to ⌊)
Equality testing (equivalent to ⌊⌊)
n^2 Superscripts and other mathematical formatting allowed

PERFORMANCE ANALYSIS:

What are the Criteria for judging algorithms that have a more direct relationship toperformance? computing time and storage requirements.

Performance evaluation can be loosely divided into two major phases:
a priori estimates and
a posteriori testing.
refer as *performance analysis* and *performance measurement* respectively

The space complexity of an algorithm is the amount of memory it needs to run tocompletion.
The time complexity of an algorithm is the amount of computer time it needs to run tocompletion.

Space Complexity:

Space Complexity Example:
Algorithm abc(a,b,c)
{
return a+b++*c+(a+b-c)/(a+b) +4.0;
}

The Space needed by each of these algorithms is seen to be the sum of the followingcomponent

A fixed part that is independent of the characteristics (eg:number,size)of the inputs andoutputs.
The part typically includes the instruction space (ie. Space for the code), space for simple variable and fixed-size component variables (also called aggregate) space for constants, and so on.

2. A variable part that consists of the space needed by component variables whose size is dependent on the particular problem instance being solved, the space needed by referencedvariables (to the extent that is depends on instance characteristics), and the recursion stackspace.

The space requirement s(p) of any algorithm p may therefore be written as,S(P) = c+ Sp(Instance characteristics)
Where 'c' is a constant.

Example 2:

Algorithm sum(a,n)
{ s=0.0;
for I=1 to n dos= s+a[I]; return s;
}
The problem instances for this algorithm are characterized by n,the number ofelements to be summed.
The space needed d by 'n' is one word, since it is of type integer.
The space needed by 'a'a is the space needed by variables of tyepe array offloating point numbers.
This is atleast 'n' words, since 'a' must be large enough to hold the 'n'elements to be summed.
So,we obtain Ssum(n)>=(n+s)
[n for a[],one each for n,I a& s]

Time Complexity:

The time T(p) taken by a program P is the sum of the compile time and therun time(execution time)

The compile time does not depend on the instance characteristics. Also we may assume that a compiled program will be run several times without recompilation .Thisrum time is denoted by tp(instance characteristics).

The number of steps any problem statement is assigned depends on the kind ofstatement.

- For example, comments à 0 steps.

Assignment statements is 1 steps.
[Which does not involve any calls to other algorithms]
Interactive statement such as for, while & repeat-untilà Control part of the statement.

We introduce a variable, count into the program statement to increment count with initial value 0.Statement to increment count by the appropriate amount are introducedinto the program.

This is done so that each time a statement in the original program is executescount is incremented by the step count of that statement.

Algorithm:

```
Algorithm sum(a,n)
{
s= 0.0;
count = count+1;for I=1 to n do
{
count =count+1;s=s+a[I]; count=count+1;
}
count=count+1; count=count+1; return s;
}
```

If the count is zero to start with, then it will be 2n+3 on termination. So eachinvocation of sum execute a total of 2n+3 steps.

The second method to determine the step count of an algorithm is to build atable in which we list the total number of steps contributes by each statement.

 First determine the number of steps per execution (s/e) of the statement and thetotal number of times (ie., frequency) each statement is executed.

By combining these two quantities, the total contribution of all statements, the step count for the entire algorithm is obtained.

Statement	*Steps perexecution*	*Frequency*	*Total*

1. Algorithm
Sum(a,n)2.{
3. S=0.0;
4. for I=1 to n do5.
s=s+a[I];
6. return s;7. }
	0
	0
	1
	1
	1
	1
	0
		-
		-1
		n+1n 1
		-
			0
			0
			1
			n+1n 1
			0

Total			2n+3

analyse an Algorithm?

Let us form an algorithm for Insertion sort (which sort a sequence of numbers).The pseudocode for the algorithm is give below.

Pseudo code for insertion Algorithm:

Identify each line of the pseudo code with symbols such as C1, C2 ..

Pseudo code for Insertion Algorithm	Line Identification
for j=2 to A length	C1
key=A[j]	C2
//Insert A[j] into sorted Array A[1.....j-1]	C3
i=j-1	C4
while i>0 & A[j]>key	C5
A[i+1]=A[i]	C6
i=i-1	C7
A[i+1]=key	C8

Let Ci be the cost of ith line. Since comment lines will not incur any cost C3=0.

Cost	No. Of times Executed
C1	N
C2	n-1
C3=0	n-1
C4	n-1
C5	$\sum_{j=2}^{n-1} t_j$
C6	$\sum_{j=2}^{n} t_j - 1$
C7	$\sum_{j=2}^{n} t_j - 1$
C8	n-1

Running time of the algorithm is:

$T(n)=C1n+C2(n-1)+0(n-1)+C4(n-1)+C5(\sum_{j=2}^{n-1} t_j)+C6(\sum_{j=2}^{n} t_j - 1)+C7(\sum_{j=2}^{n} t_j - 1)+C8(n-1)$

Best case:

It occurs when Array is sorted.All t_j values are 1.

$T(n)=C1n+C2(n-1)+0\ (n-1)+C4(n-1)+C5(\sum_{j=2}^{n} 1\ )+C6(\sum_{j=2}^{n} 1-1\ )+C7(\sum_{j=2}^{n} 1-1\ )+C8(n-1)$

$=C1n+C2\ (n-1)+0\ (n-1)+C4\ (n-1)+C5+C8\ (n-1)$

$=(C1+C2+C4+C5+C8)\ n-(C2+C4+C5+C8)$

· Which is of the form an+b.

· Linear function of n.

So, linear growth.

Worst case:

It occurs when Array is reverse sorted, and tj =j

$T(n)=C1n+C2(n-1)+0\ (n-1)+C4(n-1)+C5(\sum_{j=2}^{n} j)+C6(\sum_{j=2}^{n} j-1\ )+C7(\sum_{j=2}^{n} j-1\ )+C8(n-1)$

$=C1n+C2(n-1)+C4(n-1)+C5(\frac{n(n-1)}{2}-1)+C6(\sum_{j=2}^{n} \frac{(n-1)}{2})+C7(\sum_{j=2}^{n} (n-1))+C8(n-1)$

which is of the form an^2+bn+c

Quadratic function. So in worst case insertion set grows in n2.Why we concentrate on worst-case running time?

· The worst-case running time gives a guaranteed upper bound on the running time for any input.

· For some algorithms, the worst case occurs often. For example, when searching, theworst case often occurs when the item being searched for is not present, and searchesfor absent items may be frequent.

· Why not analyze the average case? Because it's often about as bad as the worst case.

Order of growth:

It is described by the highest degree term of the formula for running time. (Drop lower-orderterms. Ignore the constant coefficient in the leading term.)

Example: We found out that for insertion sort the worst-case running time is of the form $an^2 + bn + c$.

Drop lower-order terms. What remains is an2.Ignore constant coefficient. It results in n^2.But we cannot say that the worst-case running time T(n) equals n^2 .Rather It grows like n^2 . But itdoesn't equal n^2.We say that the running time is $\Theta(n^2)$ to capture the notion that the order of growth is n^2.

We usually consider one algorithm to be more efficient than another if its worst-caserunning time has a smaller order of growth.

Complexity of Algorithms

The complexity of an algorithm M is the function f(n) which gives the running time and/or storage space requirement of the algorithm in terms of the size 'n' of the input data. Mostly, the storage space required by an algorithm is simply a multiple of the data size 'n'.

Complexity shall refer to the running time of the algorithm.

The function f(n), gives the running time of an algorithm, depends not only on the size 'n' of the input data but also on the particular data. The complexity function f(n) for certain cases are:

1. **Best Case** : The minimum possible value of f(n) is called the best case.

2. **Average Case** : The expected value of f(n).

3. **Worst Case** : The maximum value of f(n) for any key possible input.

ASYMPTOTIC NOTATION

- **Formal way notation to speak about functions and classify them**

The following notations are commonly use notations in performance analysis and used tocharacterize the complexity of an algorithm:

1. Big–OH (O) ,
2. Big–OMEGA (Ω),
3. Big–THETA (Θ) and
4. Little–OH (o)

Asymptotic Analysis of Algorithms:

Our approach is based on the *asymptotic complexity* measure. This means that we don't try tocount the exact number of steps of a program, but how that number grows with the size of theinput to the program. That gives us a measure that will work for different operating systems, compilers and CPUs. The asymptotic complexity is written using big-O notation.

- It is a way to describe the characteristics of a function in the limit.
- It describes the rate of growth of functions.
- Focus on what's important by abstracting away low-order terms and constant factors.
- It is a way to compare "sizes" of functions:O≈ ≤

- Θ ≈ =o ≈ < ω ≈ >

.

.

Time complexity	Name	Example
O(1)	Constant	Adding an element to thefront of a linked list
O(logn)	Logarithmic	Finding an element in asorted array
O (n)	Linear	Finding an element in an unsorted array
O(nlog n)	Linear	Logarithmic Sorting n itemsby 'divide-and-conquer'-Mergesort
$O(n^2)$	Quadratic	Shortest path between two nodes in a graph
$O(n^3)$	Cubic	Matrix Multiplication
$O(2^n)$	Exponential	The Towers of Hanoiproblem

.

- **Big 'oh':** the function f(n)=O(g(n)) iff there exist positive constants c and no such thatf(n)<=c*g(n) for all n, n>= no.
- **Omega:** the function f(n)=(g(n)) iff there exist positive constants c and no such thatf(n) >= c*g(n) for all n, n >= no.
- **Theta:** the function f(n)=(g(n)) iff there exist positive constants c1,c2 and no such that c1g(n) <= f(n) <= c2 g(n) for all n, n >= no

· Big-O Notation

.

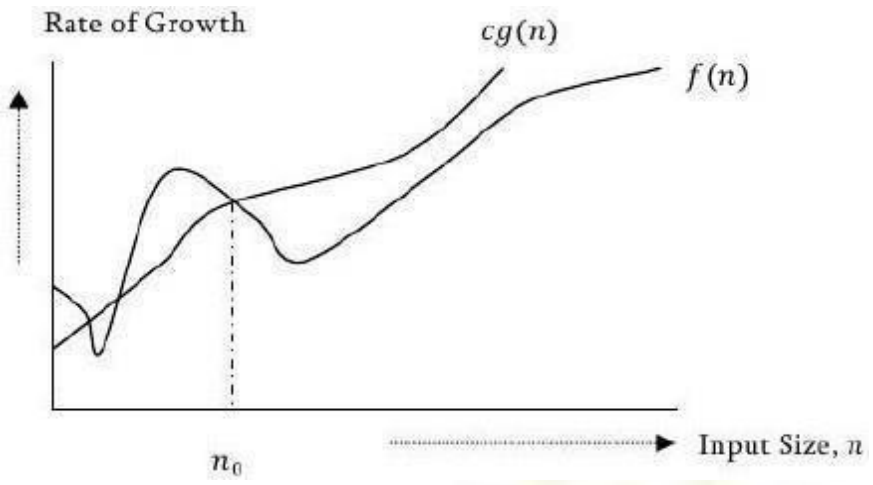
· This notation gives the tight upper bound of the given function. Generally we represent it as f(n) = O(g (11)). That means, at larger values of n, the upper bound off(n) is g(n). For example, if $f(n) = n^4 + 100n^2 + 10n + 50$ is the given algorithm, then n^4 is g(n). That means g(n) gives the maximum rate of growth for f(n) at larger values of n.

.

· **O —notation** defined as O(g(n)) = {f(n): there exist positive constants c and n_o such that 0 <= f(n) <= cg(n) for all n >= n_o}. g(n) is an asymptotic tight upper bound for f(n). Our objective is to give some rate of growth g(n) which is greater than given algorithms rate of growth f(n).

.

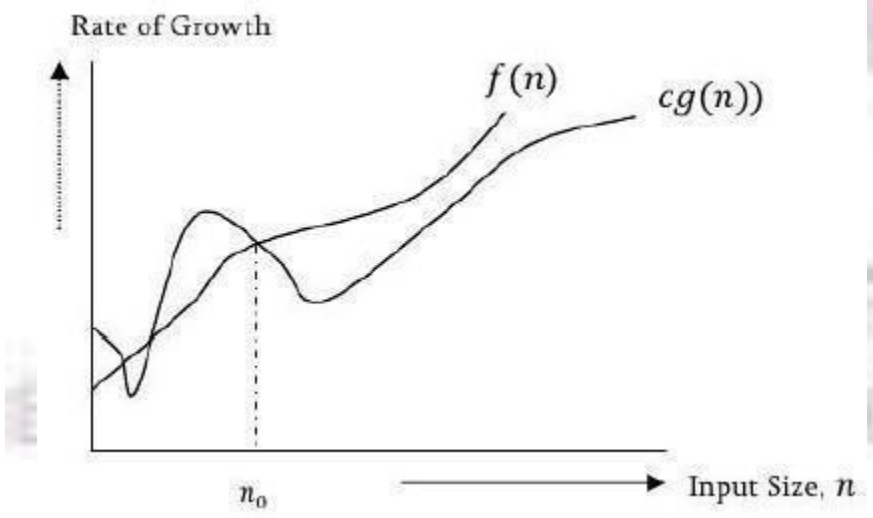
· In general, we do not consider lower values of n. That means the rate of growth at lower values of n is not important. In the below figure, n_o is the point from which we consider the rate of growths for a given algorithm. Below n_o the rate of growths may be different.



Note Analyze the algorithms at larger values of n only What this means is, below no we donot care for rates of growth.

Omega— Ω notation

Similar to above discussion, this notation gives the tighter lower bound of the givenalgorithm and we represent it as f(n) = Ω (g(n)). That means, at larger values of n, thetighter lower bound of f(n) is g
For example, if f(n) = $100n^2 + 10n + 50$, g(n) is Ω (n^2).
The . Ω. notation as be defined as Ω (g (n)) = {f(n): there exist positive constants c andn_o such that 0 <= cg (n) <= f(n) for all n >= n_o}. g(n) is an asymptotic lower bound for f(n). Ω (g (n)) is the set of functions with smaller or same order of growth as f(n).

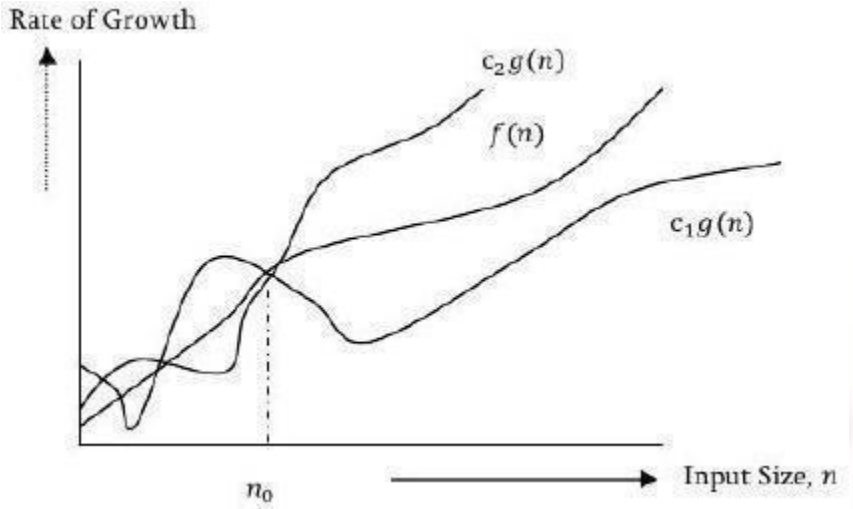


Theta- Θ notation

This notation decides whether the upper and lower bounds of a given function are same ornot. The average running time of algorithm is always between lower bound and upper bound.

If the upper bound (O) and lower bound (Ω) gives the same result then Θ notation will also have the same rate of growth. As an example, let us assume that f(n) = 10n + n is the expression. Then, its tight upper bound g(n) is O(n). The rate of growth in best case is g (n) = 0(n). In this case, rate of growths in best case and worst are same. As a result, the average case will also be same.

None: For a given function (algorithm), if the rate of growths (bounds) for O and Ω are notsame then the rate of growth Θ case may not be same.



Now consider the definition of Θ notation It is defined as Θ (g(n)) = {f(71): there exist positive constants C1, C2 and no such that O<=5 c_1g(n) <= f(n) <= c_2g(n) for all n >= n_o}.g(n) is an asymptotic tight bound for f(n). Θ (g(n)) is the set of functions with the same order of growth as g(n).

Important Notes

For analysis (best case, worst case and average) we try to give upper bound (O) and lower bound (Ω) and average running time (Θ). From the above examples, it should also be clearthat, for a given function (algorithm) getting upper bound (O) and lower bound (Ω) and average running time (Θ) may not be possible always.
For example, if we are discussing the best case of an algorithm, then we try to give upper bound (O) and lower bound (Ω) and average running time (Θ).
In the remaining chapters we generally concentrate on upper bound (O) because knowing lower bound (Ω) of an algorithm is of no practical importance and we use 9 notation if upper bound (O) and lower bound (Ω) are same.

Little Oh Notation

The little Oh is denoted as o. It is defined as : Let, f(n} and g(n} be the non negativefunctions th
Lim= $f(n)$
= 0
such that f(n}= o(g{n)} i.e f of n is little Oh of g of n.

f(n) = o(g(n)) if and only if f'(n) = o(g(n)) and f(n) != Θ {g(n))

Applications of Divide and conquer rule or algorithm:

- Binary search,
- Quick sort,
- Merge sort,
- Strassen's matrix multiplication.

Binary search or Half-interval search algorithm:

1. This algorithm finds the position of a specified input value (the search "key") within an array sorted by key value.
2. In each step, the algorithm compares the search key value with the key value of themiddle element of the array.
3. If the keys match, then a matching element has been found and its index, or position,is returned.
4. Otherwise, if the search key is less than the middle element's key, then the algorithmrepeats its action on the sub-array to the **left** of the middle element or, if the search key is greater, then the algorithm repeats on sub array to the **right** of the middle element.
5. If the search element is less than the minimum position element or greater than themaximum position element then this algorithm returns not found.

Binary search algorithm by using recursive methodology:

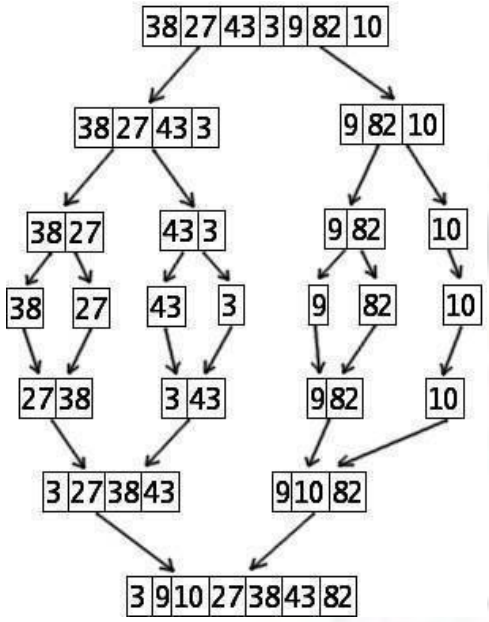
Program for binary search (recursive)	Algorithm for binary search (recursive)
int binary_search(int A[], int key, int imin, int imax)	**Algorithm binary_search(A, key, imin, imax)**
{ **if (imax < imin)** **return array is empty;if(key<imin || K>imax)** **return element not in array listelse** **{** **int imid = (imin +imax)/2;if (A[imid] > key)** **return binary_search(A, key, imin, imid-1);else if (A[imid] < key)** **return binary_search(A, key, imid+1, imax);else** **return imid;** **}** **}**	**{** **if (imax < imin) then return "array is empty";** **if(key<imin || K>imax) then return "element not in array list"else** **{** **imid = (imin +imax)/2;if (A[imid] > key) then** **return binary_search(A, key, imin, imid-1);else if (A[imid] < key) then** **return binary_search(A, key, imid+1, imax);else** **return imid;** **}** **}**

Time Complexity: Data structure:- Array

For successful search	**Unsuccessful search**
Worst case∟ O(log n) or θ(log n)Average case∟ O(log n) or θ(log n) Best case∟ O(1) or θ(1)	θ(log n):- for all cases.

MERGE SORT:

The merge sort splits the list to be sorted into two equal halves, and places them in separate arrays. This sorting method is an example of the DIVIDE-AND-CONQUER paradigm i.e. it breaks the data into two halves and then sorts the two half data sets recursively, and finally merges them to obtain the complete sorted list. The merge sort is a comparison sort and has an algorithmic complexity of $O(n \log n)$. Elementary implementations of the merge sort make use oftwo arrays - one for each half of the data set. The following image depicts the complete procedureof merge sort.



Advantages of Merge Sort:

1. Marginally faster than the heap sort for larger sets
2. Merge Sort always does lesser number of comparisons than Quick Sort. Worst case for merge sort does about 39% less comparisons against quick sort's average case.
3. Merge sort is often the best choice for sorting a linked list because the slow random- access performance of a linked list makes some other algorithms (such as quick sort) perform poorly, and others (such as heap sort) completely impossible.

NARSIMHA REDDY ENGINEERING COLLEGE

Algorithm for Merge sort:

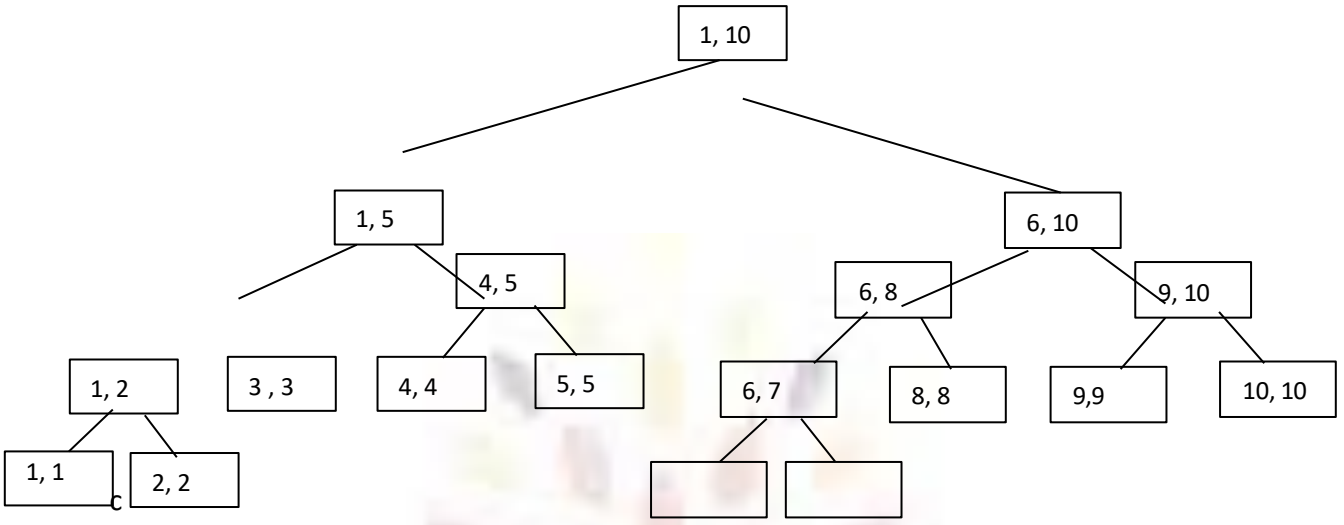
```
Algorithm mergesort(low, high)
{
if(low<high) then          // Dividing Problem into Sub-problems and
{          this “mid” is for finding where to split the set.mid=(low+high)/2;
mergesort(low,mid);
mergesort(mid+1,high); //Solve the sub-problemsMerge(low,mid,high); // Combine the solution
}
}
void Merge(low, mid,high){k=low;
i=low; j=mid+1;
while(i<=mid&&j<=high) do{if(a[i]<=a[j]) then
{
temp[k]=a[i];i++;
k++;
}
else
{
temp[k]=a[j];j++;
k++;
}
}
while(i<=mid) do{temp[k]=a[i];
i++;k++;
}

while(j<=high) do{temp[k]=a[j];
j++;k++;
}
For k=low to high doa[k]=temp[k];
}
For k:=low to high do a[k]=temp[k];
}
```

NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE

Tree call of Merge sort

Consider a example: (From text book) A[1:10]={310,285,179,652,351,423,861,254,450,520}



Tree call of Merge sort (1, 10)

Tree all of Merge Sort Represents the sequence of recursive calls that are produced by

"Once observe the explained notes in class room"

Computing Time for Merge sort:

The time for the merging operation in proportional to n, then computing time for merge sortis described by using recurrence relation.

T(n)= a if n=1; 2T(n/2)+ cn f n>1

Here c, a Constants. If n is power of 2, $n=2^k$

Form recurrence relationT(n)= 2T(n/2) + cn

2[2T(n/4)+cn/2] + cn

4T(n/4)+2cn

2^2 T(n/4)+2cn

2^3 T(n/8)+3cn

2^4 T(n/16)+4cn2^k T(1)+kcn an+cn(log n)

merge sort. By representing it by in the form of Asymptotic notation O isT(n)=O(nlog n)

QUICK SORT

Quick Sort is an algorithm based on the DIVIDE-AND-CONQUER paradigm that selects a pivotelement and reorders the given list in such a way that all elements smaller to it are on one side and those bigger than it are on the other. Then the sub lists are recursively sorted until the list getscompletely sorted. The time complexity of this algorithm is O (n log n).

- Auxiliary space used in the average case for implementing recursive function calls is O (log n) and hence proves to be a bit space costly, especially when it comes to largedata sets.
- Its worst case has a time complexity of $O(n^2)$ which can prove very fatal for largedata sets.

Competitive sorting algorithms

Algorithm for Quick sort

```
Algorithm quickSort (a, low, high) {If(high>low) then{ m=partition(a,low,high);
if(low<m) then quick(a,low,m); if(m+1<high) then quick(a,m+1,high);
}}

Algorithm partition(a, low, high){i=low,j=high;
mid=(low+high)/2;pivot=a[mid];
while(i<=j) do { while(a[i]<=pivot)
i++;
while(a[j]>pivot)j--;
if(i<=j){  temp=a[i];
a[i]=a[j]; a[j]=temp;
i++;
j--;
}}
return j;
}
```

Name	Time Complexity			Space Complexity
	Best case	AverageCase	WorstCase	
Bubble	O(n)	-	$O(n^2)$	O(n)
Insertion	O(n)	$O(n^2)$	$O(n^2)$	O(n)
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(n)
Quick	O(log n)	O(n log n)	O(n2)	O(n + log n)

Merge	O(n log n)	O(n log n)	O(n log n)	O(2n)
Heap	O(n log n)	O(n log n)	O(n log n)	O(n)

Strassen's Matrix Multiplication:

Let A and B be two n×n Matrices. The product matrix C=AB is also a n×n matrix whose i, jthelement is formed by taking elements in the ith row of A and jth column of B and multiplyingthem to get C(i, j)=$\sum_{1\le k\le n} A(i, k)B(k, j)$

Here 1≤ i & j ≤ n means i and j are in between 1 and n.

To compute C(i, j) using this formula, we need n multiplications.

The divide and conquer strategy suggests another way to compute the product of two n×nmatrices.
For Simplicity assume n is a power of 2 that is n=2^k
Here k any nonnegative integer.
If n is not power of two then enough rows and columns of zeros can be added to both A andB, so that resulting dimensions are a power of two.
Let A and B be two n×n Matrices. Imagine that A & B are each partitioned into four squaresub matrices. Each sub matrix having dimensions n/2×n/2.
The product of AB can be computed by using previous formula.If AB is product of 2×2 matrices then
If AB is product of 2×2 matrices then

$A11\ A12\ B11\ B12\ C11\ C12$ ( ) ( ) = ( )
$A21\ A22\ B21\ B22\ C21\ C22$

$C_{11}=A_{11}B_{11}+A_{12}B_{21}$ $C_{12}=A_{11}B_{12}+A_{12}B_{22}$ $C_{21}=A_{21}B_{11}+A_{22}B_{21}$ $C_{22}= A_{21}B_{12}+A_{22}B_{22}$

Here 8 multiplications and 4 additions are performed.
Note that Matrix Multiplication are more Expensive than matrix addition and subtraction.

$$T(n)= \begin{cases} b & \text{if } n\le 2; \\ 8T(n/2)+ cn^2 & \text{if } n>2 \end{cases}$$

Volker strassen has discovered a way to compute the $C_{i,j}$ of above using 7 multiplications and 18 additions or subtractions.
For this first compute 7 n/2×n/2 matrices P, Q, R, S, T, U & VP=(A11+A22)(B11+B22)
Q=(A21+A22)B11 R=A11(B12-B22) S=A22(B21-B11) T=(A11+A12)B22 U=(A21-A11)(B11+B12)
V=(A12-A22)(B21+B22)

C11=P+S-T+V
C12=R+T
C21=Q+S
C22=P+R-Q+U

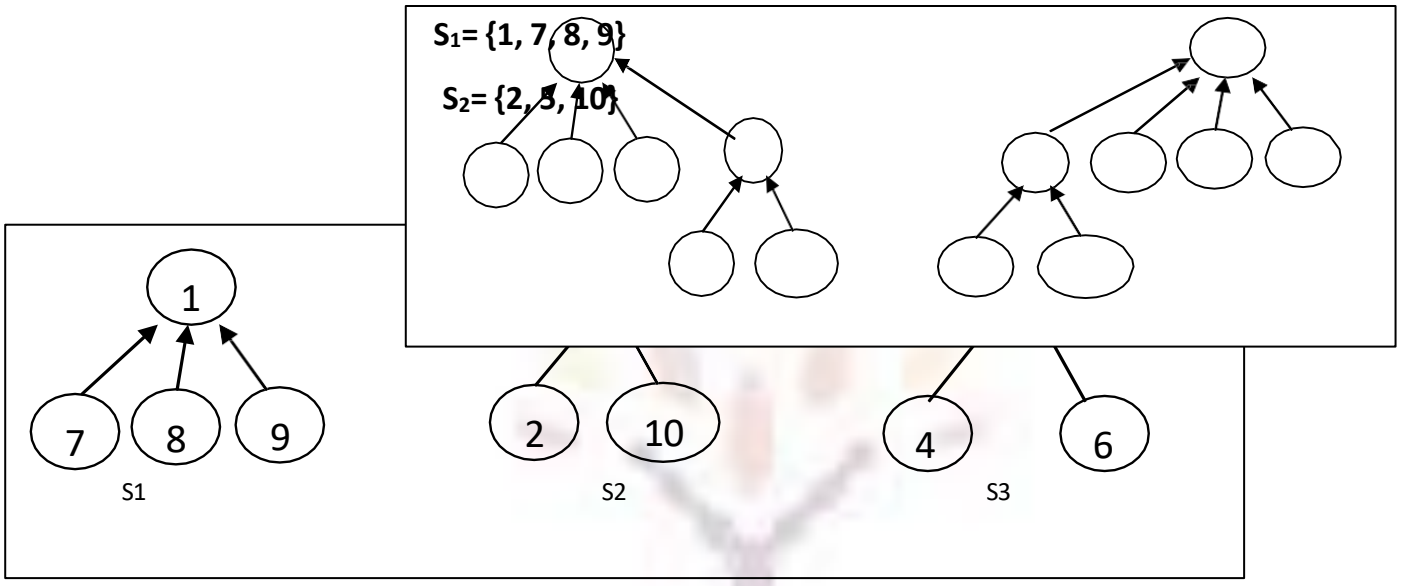
$$T(n)= \begin{cases} b & \text{if } n\le 2; \\ 7T(n/2)+ cn^2 & \text{if } n>2 \end{cases}$$

UNIT-II

Disjoint Sets: Disjoint set operations, union and find algorithms
Backtracking: General method, applications, n-queen's problem, sum of subsets problem,graph coloring

Disjoint Sets: If S_i and S_j, $i \neq j$ are two sets, then there is no element that is in both S_i and S_j.

For example: n=10 elements can be partitioned into three disjoint sets,

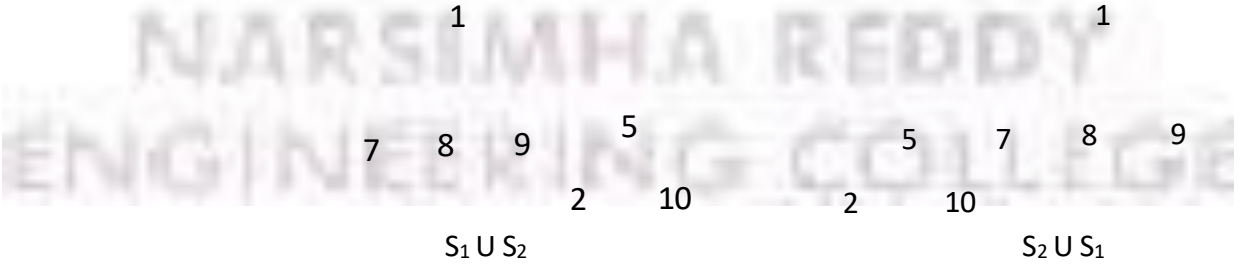


Disjoint set Operations:

- Disjoint set Union
- Find(i)

Disjoint set Union: Means $S_1 \cup S_2$ combination of two disjoint sets elements. Form aboveexample $S_1 \cup S_2 = \{1,7,8,9,5,2,10\}$

For $S_1 \cup S_2$ tree representation, simply make one of the tree is a subtreeof the other.



$S_1 \cup S_2$ $S_2 \cup S_1$

Find: Given element i, find the set containing i.

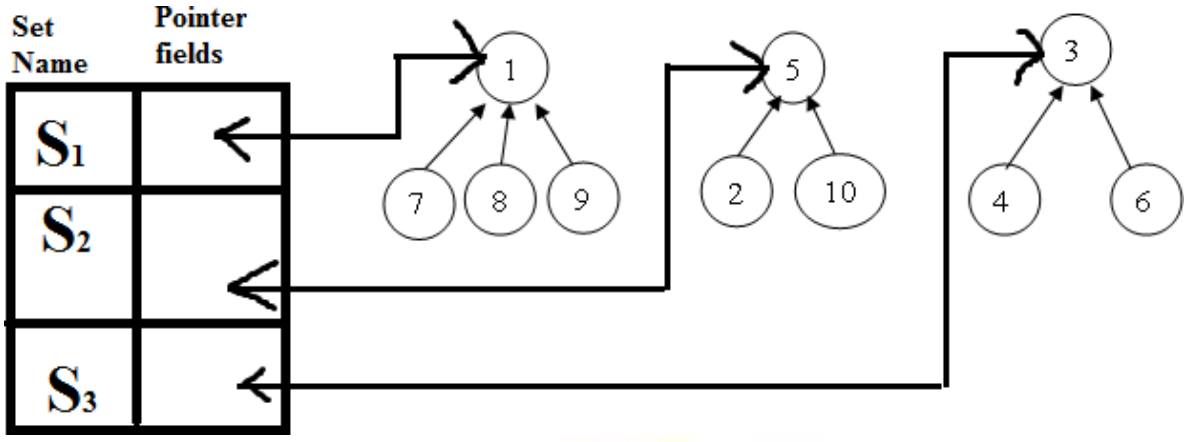
Form above example:

Find(4) S_3 Find(1)

S_1 Find(10) S_2

DATA REPRESENTATION OF SETS:

Tress can be accomplished easily if, with each set name, we keep a pointer tothe root of thetree representing that set.



For presenting the union and find algorithms, we ignore the set names andidentify sets justby the roots of the trees representing them.
For example: if we determine that element 'i' is in a tree with root 'j' has apointer to entry'k' in the set name table, then the set name is just **name[k]**

For unite (**adding or combine**) to a particular set we use FindPointer function.
Example: If you wish to unite to S_i and S_j then we wish to unite the tree with rootsFindPointer (S_i) and FindPointer (S_j)
FindPointer is a function that takes a set name and determines the root ofthe tree thatrepresents it.
For determining operations:
Find(i) 1st determine the root of the tree and find its pointer to entry in setname table.Union(i, j) Means union of two trees whose roots are i and j.

If set contains numbers 1 through n, we represents tree node
P[1:N].
Maximum number of elements.

i	1	2	3	4	5	6	7	8	9	10
P	-1	5	-1	3	-1	3	1	1	1	5

-1.

nd(i) by following the indices, starting at i until we reach a node with parent value
Example: Find(6) start at 6 and then moves to 6's parent. Since P[3] is negative, we reachedthe root.

Algorithm for finding Union(i, j):	Algorithm for find(i)

Algorithm Simple union(i, j)
{
P[i]:=j; // Accomplishes the union
}
	Algorithm SimpleFind(i)
	{
	While(P[i]≥0) do i:=P[i];
	return i;
	}

If n numbers of roots are there then the above algorithms are not useful forunion and find.For union of n trees Union(1,2), Union(2,3), Union(3,4),…..Union(n-1,n).
For Find i in n trees Find(1), Find(2),….Find(n).

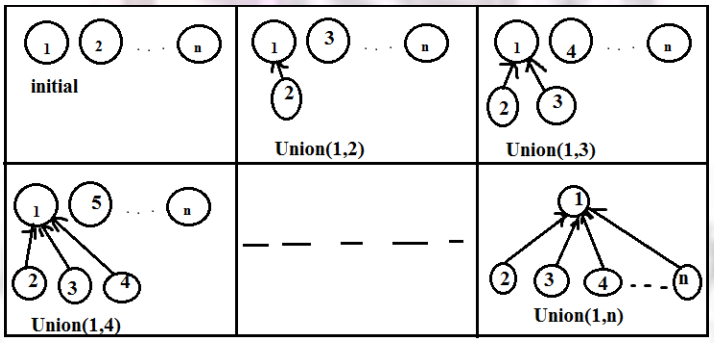
Time taken for the union (simple union) is O(1) (constant).
For the n-1 unions O(n).

Time taken for the find for an element at level i of a tree is O(i).
For n finds $O(n^2)$.

To improve the performance of our union and find algorithms by avoiding thecreation ofdegenerate trees. For this we use a weighting rule for union(i, j)

WEIGHTING RULE FOR UNION(I, J):

If the number of nodes in the tree with root 'i' is less than the tree with root 'j',then make 'j'the parent of 'i'; otherwise make 'i' the parent of 'j'.



Tree obtained using the weighting rule

Algorithm for weightedUnion(i, j)

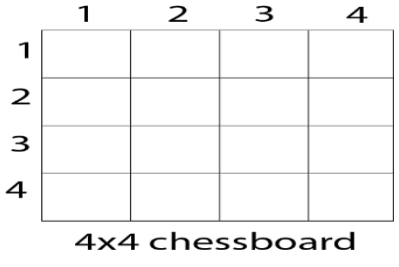
```
Algorithm WeightedUnion(i,j)
//Union sets with roots i and j, i≠j
// The weighting rule, p[i]= -count[i] and p[j]= -count[j].
{
temp := p[i]+p[j];
if (p[i]>p[j]) then
{ // i has fewer
nodes.P[i]:=j;
P[j]:=temp;
}
else
{ // j has fewer or equal
nodes.P[j] := i;
P[i] := temp;
}
```

N-QUEENS PROBLEM

N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queensattack each other by being in the same row, column or diagonal.

It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3.So first we will consider the 4 queens problem and then generate it to n - queens problem.

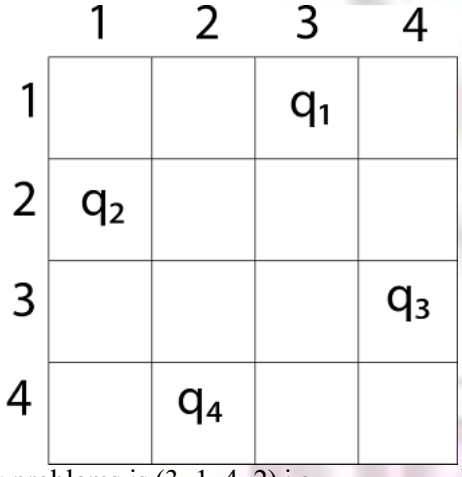
Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.



4x4 chessboard

Since, we have to place 4 queens such as q_1 q_2 q_3 and q_4 on the chessboard, such that no two queens attack each other. In such a conditional each queen must be placed on a different row, i.e., we put queen"i" on row "i."
Now, we place queen q_1 in the very first acceptable position (1, 1). Next, we put queen q_2 so that both

these queens do not attack each other. We find that if we place q_2 in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for q_2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q_3' safely. So we backtrack one step and place the queen 'q_2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q_3' which is (3, 2). But later this positionalso leads to a dead end, and no place is found where 'q_4' can be placed safely. Then we have to backtrack till 'q_1' and place it to (1, 2) and then all other queens are placed safely by moving q_2 to (2, 4),q_3 to (3, 1) and q_4 to (4, 3). That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem. For another possible solution, the whole method is repeated for all partial solutions. The other solutions for



4 - queens problems is (3, 1, 4, 2) i.e.

The implicit tree for 4 - queen problem for a solution (2, 4, 1, 3) is as follows:

NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE

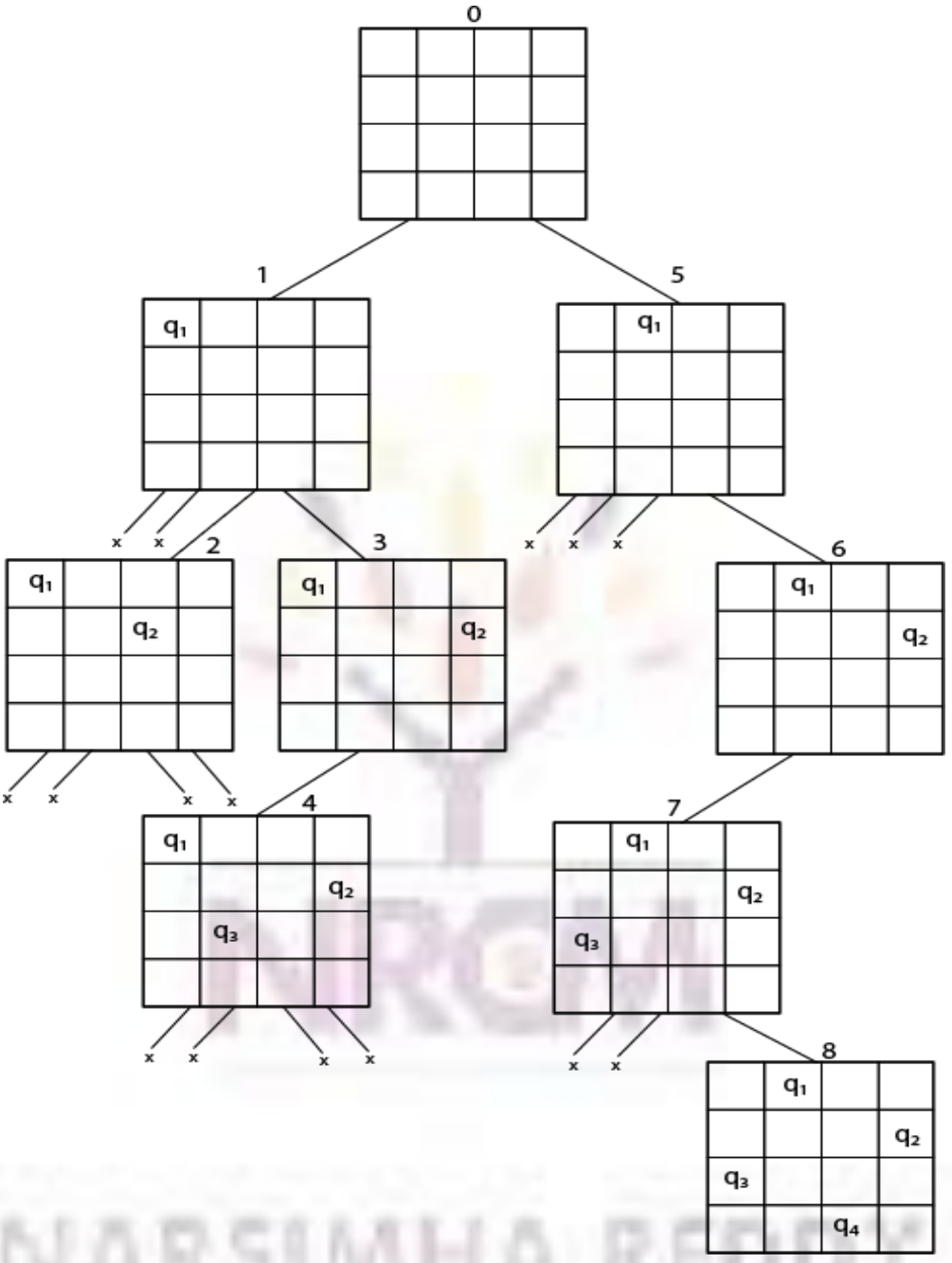
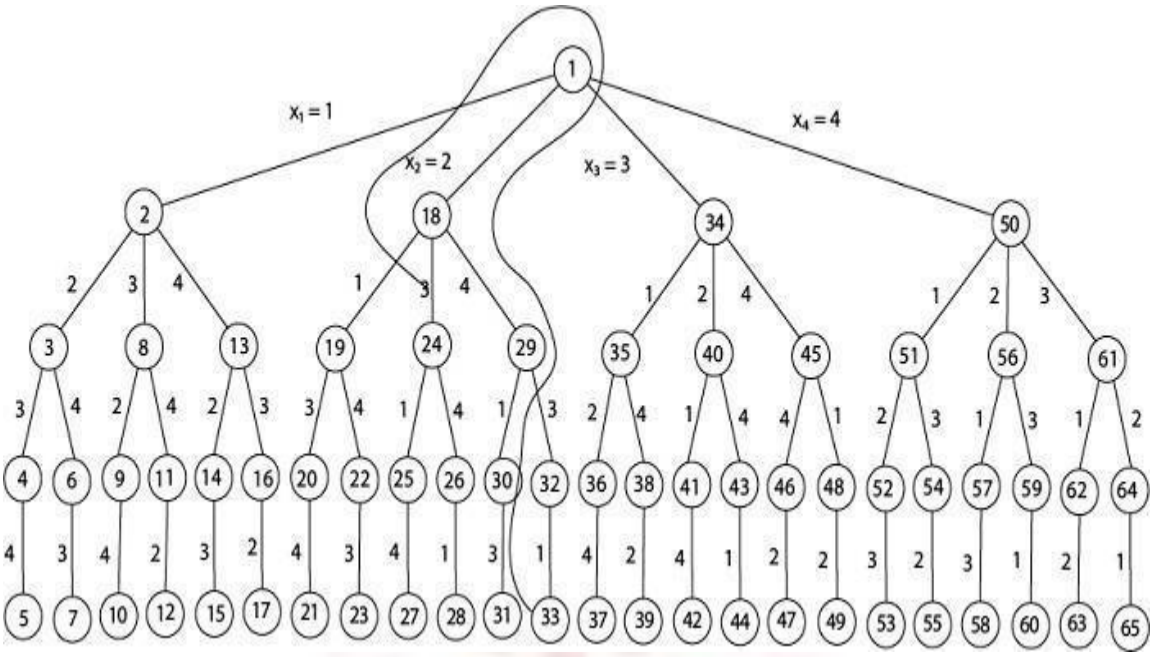


Fig shows the complete state space for 4 - queens problem. But we can use backtracking method to generate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.



ALGORITHM:

```
Place (k, i)
  {
   For j ← 1 to k - 1
    do if (x [j] = i)
     or (Abs x [j]) -
   i) = (Abs (j - k))
   then return
   false;
   RETURN TRUE;
}
```

N - QUEENS (K, N)

```
{
  For i ← 1 to n
      do if Place (k, i) then
  {
    x [k] ← i;
if (k ==n) then write (x [1        n));
else
N - Queens (k + 1, n);
}
}
```

SUBSET SUM PROBLEM

It is one of the most important problems in complexity theory. The problem is given an A set of integersa1, a2,…., an upto n integers. The question arises that is there a non-empty subset such that the sum of the subset is given as M integer?. For example, the set is given as [5, 2, 1, 3, 9],

and the sum of the subset is 9; the answer is YES as the sum of the subset [5, 3, 1] is equal to 9. This is an NP-complete problem again. It is the special case of knapsack

Let's understand this problem through

an example.problem.

We have a set of 5 integers given below:

N = 4, -2, 2, 3, 1

We want to find out the subset whose sum is equal to 5. There are many solutions to this problem.

The naïve approach, i.e., brute-force search generates all the possible subsets of the original array, i.e., there are 2n possible states. Here the running time complexity would be exponential. Then, we consider all these subsets in O(N) linear running time and checks whether the sum of the items is M or not.

The dynamic programming has pseudo-polynomial running time.

Statement: Given a set of positive integers, and a value sum, determine that the sum of the subset of a given set is equal to the given sum.

THERE ARE TWO WAYS OF SOLVING THE SUBSET PROBLEM:

- Recursion
- Dynamic programming

METHOD 1: RECURSION

Let's understand that how can we solve the problem using recursion. Consider the array which is given below:

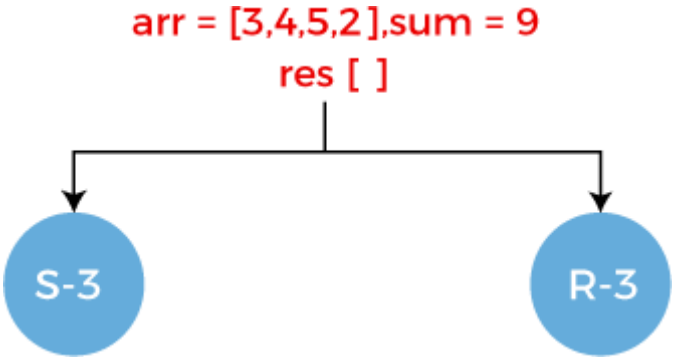
ARR = [3, 4, 5, 2]

sum = 9 result = []

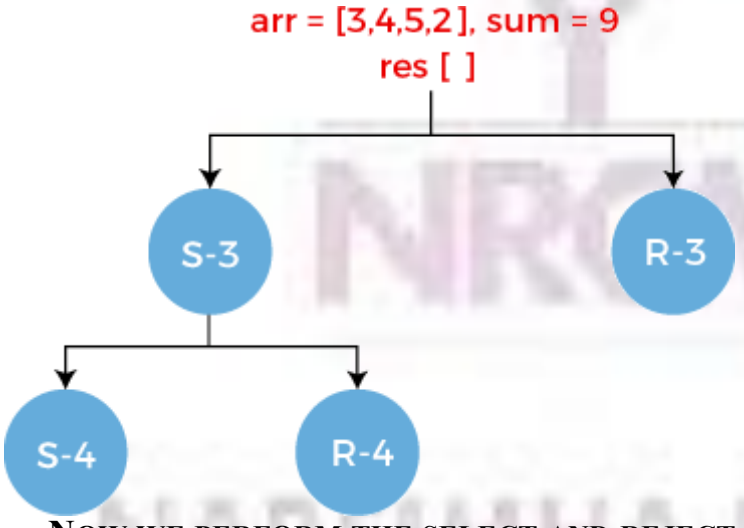
In the above example, we have taken an array, and the empty array named result that stores all the values whose resultant sum is equal to 9.

FIRST ELEMENT IN AN ARRAY IS 3. THERE ARE TWO SCENARIOS:

- First scenario is select. The sum is equal to the **target sum - value of first element, i.e.,** 9 - 3 =6 and the first element, i.e., 3 gets stored in the result array, i.e., **result[]**.
- Second scenario is reject. The array arr contains the elements 4, 5, 2, i.e., arr = [4, 5, 2] and sumwould be same as 9 as we are rejecting the element 3. The **result[]** array would remain empty.
-

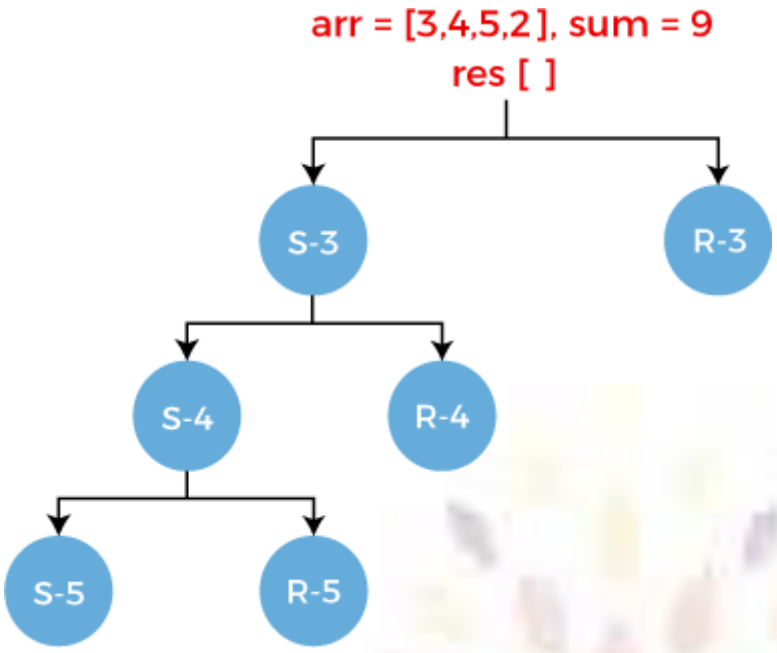


- NOW WE PERFORM THE SAME SELECT AND REJECT OPERATION ON ELEMENT 4 AS IT IS THE FIRST ELEMENT OF THE ARRAY NOW.
- Select the element 4 from the array. Since we are selecting 4 from the array so array arr would contain the elements 5, 2, i.e., arr = [5, 2]. The sum is equal to the 6-4 = 2 and the element 4 getsstored in the result arr. The result[] = {3, 4}.
- Reject the element 4 from the array. Since we are rejecting the 4 from the array so array arr would contain the elements 5, 2, i.e., arr = [5, 2]. The sum would remain same as 6 and the result array would be same as previous, i.e., {3}.



NOW WE PERFORM THE SELECT AND REJECT OPERATION ON ELEMENT 5.

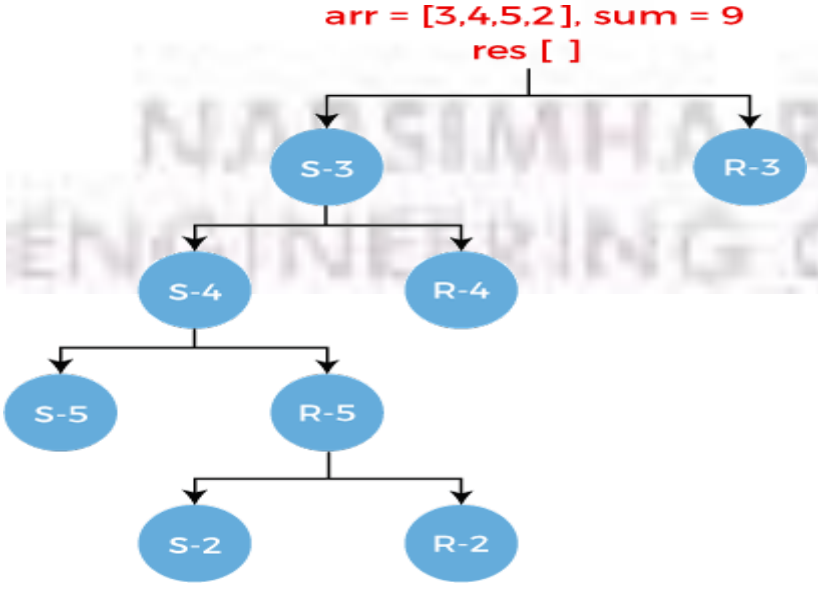
- Select the element 5 from the array. Since we are selecting 5 from the array so array arr would contain the elements 2, i.e., arr = [2]. The sum is equal to the 2 - 5 equals to -3 and the element 5gets stored in the result arr. The result[] = {3, 4, 5}.
- Reject the element 5 from the array. Since we are rejecting 5 from the array so array arr would contain the element 2, i.e., arr = [2]. The sum would remain same as previous, i.e., 6 and the result array would be same as previous, i.e., {3, 4}.



If we observe S-5, we can see that the sum is negative that returns false. It means that there is no furthersubset available in the set.

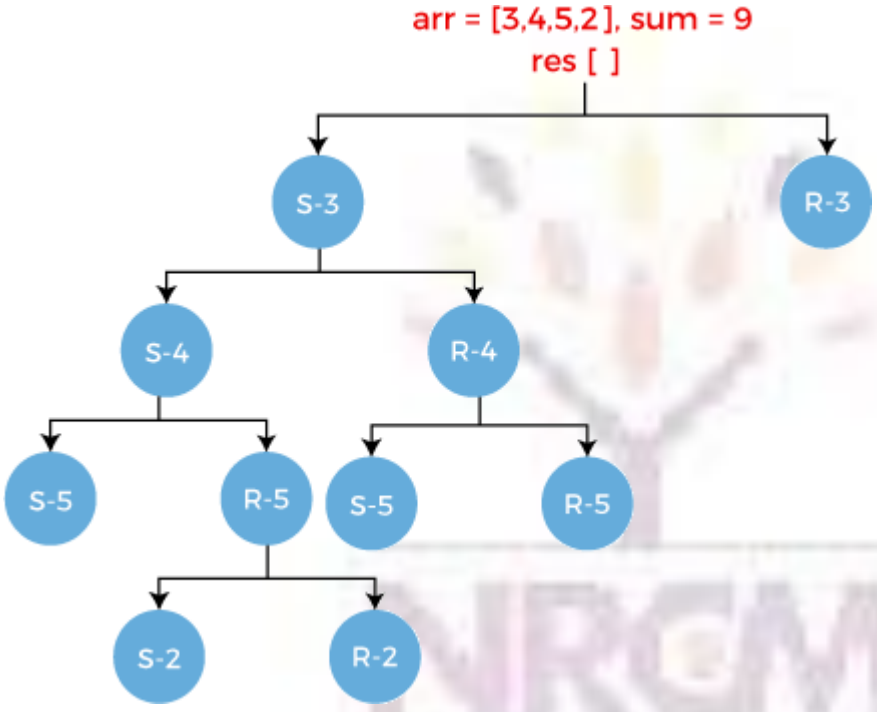
CONSIDER R-5. IT ALSO HAS TWO SCENARIOS:

- Select the element 2 from the array. Once the element 2 gets selected, the array becomes empty, i.e., arr[] = " ". The sum would be 2-2 equals to 0 and the element 2 gets stored in the result array. The result[] = [3, 4, 2].
- Reject the element 2 from the array. Once the element 2 gets rejected, the array becomes empty, i.e., arr[] = " ". The sum would be same as previous, i.e., 2 and the result array would also be same as previous, i.e., [3, 4].



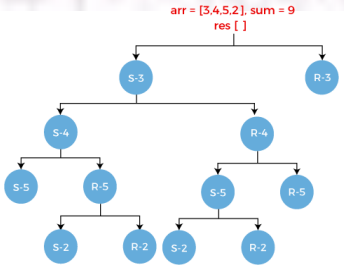
CONSIDER R-4. IT HAS TWO SCENARIOS:

- Select the element 5 from the array. Since we are selecting 5 from the array so array arr would contain the elements 2, i.e., arr = [2]. The sum would be 6-5 equals to 1 and the element 5 gets stored in the result array. The result[] = [3, 5].
- Reject the element 5 from the array. Since we are rejecting 5 from the array so array arr would contain the element 2, i.e., arr = [2]. The sum would remain same as previous, i.e., 6 and the result array would be same as previous, i.e., {3}.



CONSIDER S-5. IT HAS TWO SCENARIOS:

- Select the element 2 from the array. Since we are selecting 2 from the array so array arr wouldbe empty, i.e., arr = " ". The sum would be 1-2 equals to -1 and the element 2 gets stored in the result array. The result[] = [3, 5, 2].
- Reject the element 2 from the array. Since we are rejecting 2 from the array so array arr would become empty. The sum would remain same as previous, i.e., 1 and the result array would be same as previous, i.e., {3, 5}.

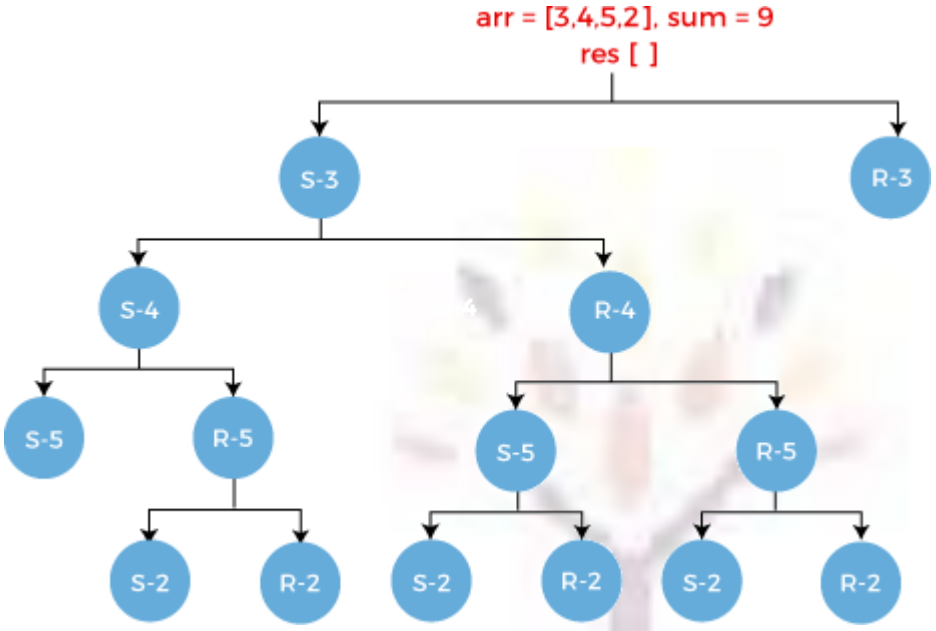


CONSIDER R-5. IT HAS TWO SCENARIOS:

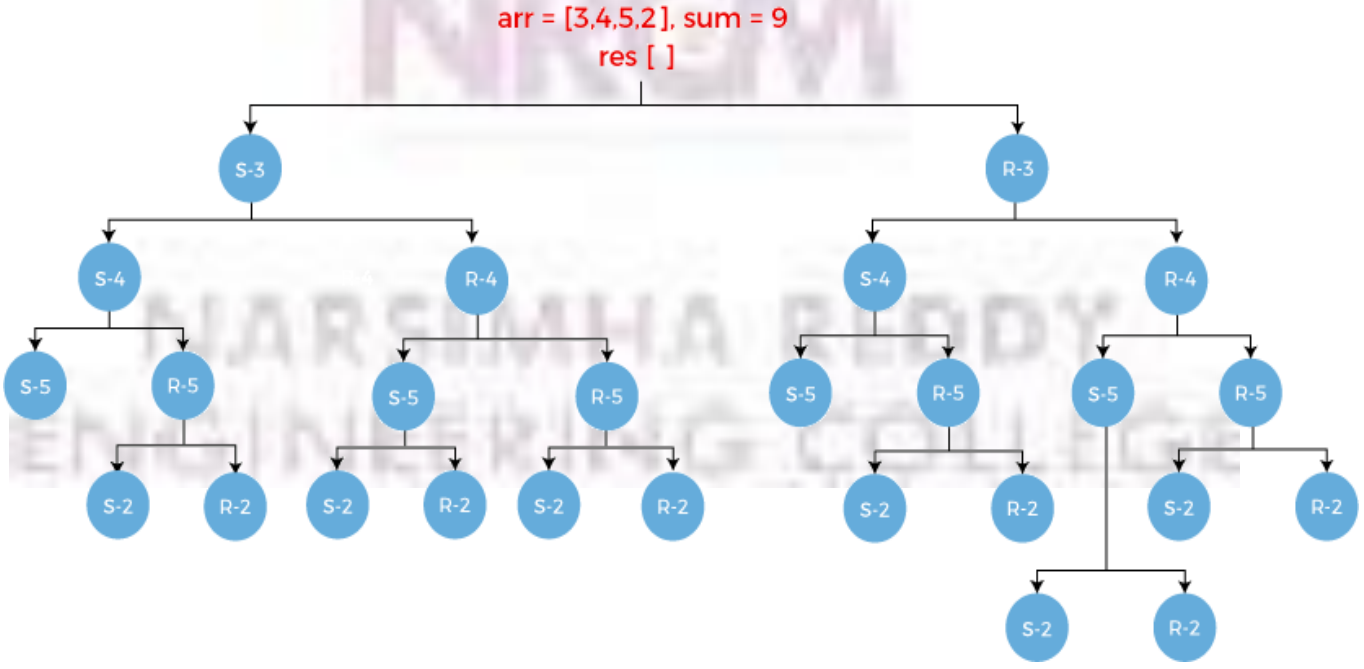
- Select the element 2 from the array. Since we are selecting 2 from the array so

array arr wouldbe empty, i.e., arr = " ". The sum would be 6-2 equals to 4 and the element 2 gets stored in the result array. The result[] = [3, 2].

- Reject the element 2 from the array. Since we are rejecting 2 from the array so array arr would become empty. The sum would remain same as previous, i.e., 6 and the result array would be same as previous, i.e., {3}.



Similarly, we get the reject case, i.e., R-3 as shown as below:



Algorithm:

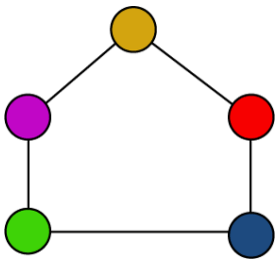
```
def sum_subset(arr, res, sum)if sum ==0
return trueif sum < 0 return false
if len(arr) == 0 and sum!= 0return false
arr.pop(0);
if len(arr) > 0 res.append(arr[0])
select = sum_subset (arr, sum-arr[0], res)reject = sum_subset (arr, res, sum)
return reject or sum
```

Graph coloring

Graph coloring can be described as a process of assigning colors to the vertices of a graph. In this, the same color should not be used to fill the two adjacent vertices. We can also call graph coloring as Vertex Coloring. In graph coloring, we have to take care that a graph must not contain any edge whose end vertices are colored by the same color. This type of graph is known as the Properly colored graph.

Example of Graph coloring

In this graph, we are showing the properly colored graph, which is described as follows:



The above graph contains some points, which are described as follows:

- The same color cannot be used to color the two adjacent vertices.
- Hence, we can call it as a properly colored graph.

APPLICATIONS OF GRAPH COLORING

There are various applications of graph coloring. Some of their important applications are described as follows:

- Assignment
- Map coloring
- Scheduling the tasks
- Sudoku
- Prepare time table
- Conflict resolution

GREEDY ALGORITHM

There are various steps to solve the greedy algorithm, which are described as follows:

Step 1: In the first step, we will color the first vertex with first color.

Step 2: Now, we will one by one consider all the remaining vertices (V -1) and do the following:

- We will color the currently picked vertex with the help of lowest number color if and only if thesame color is not used to color any of its adjacent vertices.
- If its adjacent vertices are using it, then we will select the next least numbered color.
- If we have already used all the previous colors, then a new color will be used to fill or assign tothe currently picked vertex.

UNIT-III

Dynamic Programming: General method, applications- Optimal binary search trees, 0/1 knapsack problem, All pairs shortest path problem, Traveling sales person problem, Reliability design.

Optimal Binary Search Tree

We know the key values of each node in the tree, and we also know the frequencies of each node in terms of searching means how much time is required to search a node. The frequency and key-value determine the overall cost of searching a node. The cost of searching is a very important factor in various applications. The overall cost of searching a node should be less. The time required to search a node in BST is more than the balanced binary search tree as a balanced binary search tree contains a lesser number of levels than the BST. There is one way that can reduce the cost of a binary search tree is known as an **optimal binary search tree**.
The Formula for calculating the number of trees:

$$\frac{{}^{2n}C_n}{n+1}$$

Dynamic Approach

Consider the below table, which contains the keys and frequencies.

	1	2	3	4
Keys →	10	20	30	40
Frequency →	4	2	6	3

i \ i	0	1	2	3	4
0					
1					
2					
3					
4	32				

FIRST, WE WILL CALCULATE THE VALUES WHERE J-I IS EQUAL TO ZERO.

When i=0, j=0, then j-i

= 0 When i = 1, j=1,

then j-i = 0When i = 2,

j=2, then j-i = 0When i

= 3, j=3, then j-i = 0

When i = 4, j=4, then

j-i = 0

Therefore, c[0, 0] = 0, c[1 , 1] = 0, c[2,2] = 0, c[3,3] = 0, c[4,4] = 0

NOW WE WILL CALCULATE THE VALUES WHERE J-I EQUAL TO 1.

When j=1, i=0 then j-i = 1
When j=2, i=1 then

j-i = 1When j=3, i=2

then j-i = 1 When

j=4, i=3 then j-i = 1

Now to calculate the cost, we will consider only the jth value.

The cost of c[0,1] is 4 (The key is 10, and the cost corresponding to key 10 is

4).The cost of c[1,2] is 2 (The key is 20, and the cost corresponding to key 20 is

2).The cost of c[2,3] is 6 (The key is 30, and the cost corresponding to key 30 is

6)The cost of c[3,4] is 3 (The key is 40, and the cost corresponding to key 40 is

3)

	0	1	2	3	4
0	0	4			
1		0	2		
2			0	6	
3				0	3
4					0

NOW WE WILL CALCULATE THE VALUES WHERE J-I = 2

When j=2, i=0 then

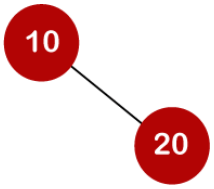
j-i = 2When j=3, i=1

then j-i = 2 When

j=4, i=2 then j-i = 2

In this case, we will consider two keys.

- When i=0 and j=2, then keys 10 and 20. There are two possible trees that can be made out from these two keys shown below:



In the first binary tree, cost would be: 4*1 + 2*2 = 8

In the second binary tree, cost would be: 4*2 + 2*1 = 10
The minimum cost is 8; therefore, c[0,2] = 8

	0	1	2	3	4
0	0	4	8		
1		0	2		
2			0	6	
3				0	3
4					0

- When i=1 and j=3, then keys 20 and 30. There are two possible trees that can be madeout from these two keys shown below:

In the first binary tree, cost would be: 1*2 + 2*6 = 14

In the second binary tree, cost would be: 1*6 + 2*2 =

10The minimum cost is 10; therefore, c[1,3] = 10

- When i=2 and j=4, we will consider the keys at 3 and 4, i.e., 30 and 40. There are twopossible trees that can be made out from these two keys shown as

below:

In the first binary tree, cost would be: 1*6 + 2*3 = 12

In the second binary tree, cost would be: 1*3 + 2*6 =

i \ 1	0	1	2	3	4
0	0	4	8^1		
1		0	2	10^3	
2			0	6	12^3
3				0	3
4					0

15The minimum cost is 12, therefore, c[2,4] = 12

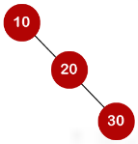
Now we will calculate the values when j-i

= 3When j=3, i=0 then j-i = 3

When j=4, i=1 then j-i = 3

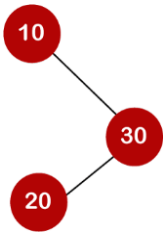
- When i=0, j=3 then we will consider three keys, i.e., 10, 20, and 30.

The following are the trees that can be made if 10 is considered as a root node.



In the above tree, 10 is the root node, 20 is the right child of node 10, and 30 is the right child ofnode 20.

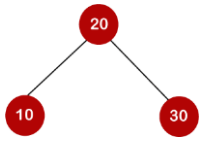
Cost would be: 1*4 + 2*2 + 3*6 = 26



In the above tree, 10 is the root node, 30 is the right child of node 10, and 20 is the left child ofnode 20.

Cost would be: 1*4 + 2*6 + 3*2 = 22

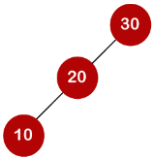
The following tree can be created if 20 is considered as the root node.



In the above tree, 20 is the root node, 30 is the right child of node 20, and 10 is the left child ofnode 20.

Cost would be: 1*2 + 4*2 + 6*2 = 22

The following are the trees that can be created if 30 is considered as the root node.



In the above tree, 30 is the root node, 20 is the left child of node 30, and 10 is the left child of node 20.
Cost would be: 1*6 + 2*2 + 3*4 = 22

In the above tree, 30 is the root node, 10 is the left child of node 30 and 20 is the right child ofnode 10.

Cost would be: 1*6 + 2*4 + 3*2 = 20

Therefore, the minimum cost is 20 which is the 3rd root. So, c[0,3] is equal to 20.

- When i=1 and j=4 then we will consider the keys 20,

30, 40c[1,4] = min{ c[1,1] + c[2,4], c[1,2] + c[3,4], c[1,3] +

c[4,4] } + 11

= min{0+12, 2+3, 10+0}+ 11

= min{12, 5, 10} + 11

The minimum value is 5; therefore, c[1,4] = 5+11 = 16

i \ j	0	1	2	3	4
0	0	4	8^1	20^3	
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

NOW WE WILL CALCULATE THE VALUES WHEN J-I = 4

When j=4 and i=0 then j-i = 4

In this case, we will consider four keys, i.e., 10, 20, 30 and 40. The frequencies of 10, 20, 30 and 40 are 4, 2, 6 and 3 respectively.

w[0, 4] = 4 + 2 + 6 + 3 = 15

If we consider 10 as the root node

then C[0, 4] = min {c[0,0] +

c[1,4]}+ w[0,4]

min {0 + 16} + 15= 31

If we consider 20 as the root node

thenC[0,4] = min{c[0,1] + c[2,4]}

+ w[0,4]

= min{4 + 12} + 15

= 16 + 15 = 31

If we consider 30 as the root node

then,C[0,4] = min{c[0,2] + c[3,4]}

+w[0,4]

= min {8 + 3} + 15

= 26

If we consider 40 as the root node

then,C[0,4] = min{c[0,3] + c[4,4]}

+ w[0,4]

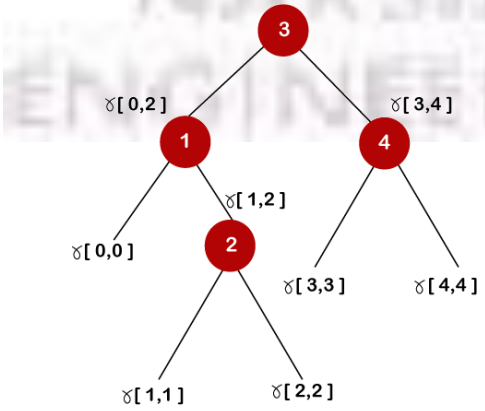
= min{20 + 0} + 15

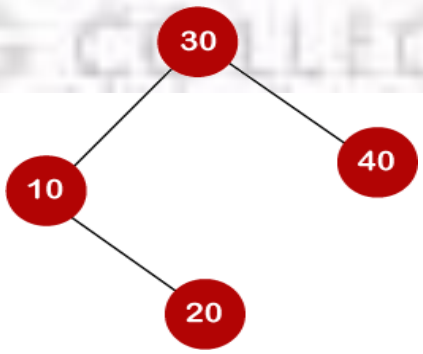
= 35

In the above cases, we have observed that 26 is the minimum cost; therefore, c[0,4] is equal to 26.

i \ j	0	1	2	3	4
0	0	4	8^1	20^3	26^3
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

The optimal binary tree can be created as:





GENERAL FORMULA FOR CALCULATING THE MINIMUM COST IS:

$C[i,j] = \min\{c[i, k-1] + c[k,j]\} + w(i,j)$

ALGORITHM:

Algorithm OBST(p, q, n)

// e[1…n+1, 0…n] : Optimal sub tree

// w[1…n+1, 0…n] : Sum of probability

// root[1…n, 1…n] : Used to construct OBST

for i ← 1 to n + 1

DO

e[i, i – 1] ← qi – 1

w[i, i – 1] ← qi – 1

END

for m ← 1 to n **do**

for i ← 1 to n – m + 1 **do**

j ← i + m – 1

e[i, j] ← ∞

w[i, j] ← w[i, j – 1] + pj + qj

for r ← i to j **do**

t ← e[i, r – 1] + e[r + 1, j] + w[i, j]

if t < e[i, j] **then**

e[i, j] ← t

root[i, j] ← r

E

N

D

E

N

D

E

N

D

end

return (e, root)

COMPLEXITY ANALYSIS OF OPTIMAL BINARY SEARCH TREE

It is very simple to derive the complexity of this approach from the above algorithm. It uses three nested loops. Statements in the innermost loop run in Q(1) time. The running time of the algorithm is computed as

$$T(n) = \sum_{m=1}^{n} \sum_{i=1}^{n-m+1} \sum_{j=i}^{n-1+1} \Theta(1)$$

$$= \sum_{m=1}^{n} \sum_{i=1}^{n-m+1} n = \sum_{m=1}^{n} n^2$$

$$= \Theta(n^3)$$

0/1 Knapsack problem

Here knapsack is like a container or a bag. Suppose we have given some items which have some weights or profits. We have to put some items in the knapsack in such a way total value producesa maximum profit.

For example, the weight of the container is 20 kg. We have to select the items in such a way that the sum of the weight of items should be either smaller than or equal to the weight of the container, and the profit should be maximum.

Consider the problem having weights and profits are:
Weights: {3, 4, 6, 5}

Profits: {2, 3, 1, 4}

The weight of the knapsack is

8 kgThe number of items is 4

The above problem can be solved by using the following method:

x_i = {1, 0, 0, 1}

= {0, 0, 0, 1}

= {0, 1, 0, 1}

The above are the possible combinations. 1 denotes that the item is completely picked and 0 means that no item is picked. Since there are 4 items so possible combinations will be:

2^4 = 16; So. There are 16 possible combinations that can be made by using the above problem. Once all the combinations are made, we have to select the combination that provides the maximum profit.

Another approach to solve the problem is dynamic programming approach. In dynamic programming approach, the complicated problem is divided into sub-problems, then we find the solution of a sub-problem and the solution of the sub-problem will be used to find the solution ofa complex problem.

First,

we create a matrix shown as below:

	0	1	2	3	4	5	6	7	8
0									
1									
2									
3									

4	

In the above matrix, columns represent the weight, i.e., 8. The rows represent the profits and weights of items. Here we have not taken the weight 8 directly, problem is divided into sub-problems, i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8. The solution of the sub-problems would be saved in the cells and answer to the problem would be stored in the final cell. First, we write the weights in the ascending order and profits according to their weights shown as below:
Start filling the table row wise top to bottom from left to right using the formula-

$$T(I, J) = \text{MAX}\{ T(I-1, J), \text{VALUE}_I + T(I-1, J - \text{WEIGHT}_I) \}$$

wi = {3, 4, 5, 6}

$P_I = \{2, 3, 4, 1\}$

The first row and the first column would be 0 as there is no item for w=0

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0								
2	0								
3	0								
4	0								

NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE

WHEN I=1, W=1

$w_1 = 3$; Since we have only one item in the set having weight 3, but the capacity of the knapsack is 1. We cannot fill the item of 3kg in the knapsack of capacity 1 kg so add 0 at M[1][1] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0							
2	0								
3	0								
4	0								

WHEN I = 1, W = 2

w_1 = 3; Since we have only one item in the set having weight 3, but the capacity of the knapsack is 2. We cannot fill the item of 3kg in the knapsack of capacity 2 kg so add 0 at M[1][2] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0						
2	0								
3	0								
4	0								

WHEN I=1, W=3

w_1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is also 3; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][3] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2					
2	0								
3	0								
4	0								

WHEN I=1, W = 4

w_1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 4; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][4] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2				
2	0								
3	0								
4	0								

WHEN I=1, W = 5

w_1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 5; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][5] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2			
2	0								
3	0								
4	0								

WHEN I =1, W=6

w_1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 6; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][6] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2		
2	0								
3	0								
4	0								

WHEN I=1, W = 7

w_1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 7; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][7] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	
2	0								
3	0								
4	0								

WHEN I =1, W =8

w_1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 8; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][8] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0								
3	0								
4	0								

NOW THE VALUE OF 'I' GETS INCREMENTED, AND

BECOMES 2.

WHEN I =2, W = 1

The weight corresponding to the value 2 is 4, i.e., $w_2 = 4$. Since we have only one item in the set having weight equal to 4, and the weight of the knapsack is 1. We cannot put the item of weight4 in a knapsack, so we add 0 at M[2][1] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0							
3	0								
4	0								

WHEN I =2, W = 2

The weight corresponding to the value 2 is 4, i.e., $w_2 = 4$. Since we have only one item in the set having weight equal to 4, and the weight of the knapsack is 2. We cannot put the item of weight4 in a knapsack, so we add 0 at M[2][2] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0						
3	0								

WHEN I =2, W = 3

The weight corresponding to the value 2 is 4, i.e., $w_2 = 4$. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 3. We can put the item of weight 3 in a knapsack, so we add 2 at M[2][3] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2					
3	0								
4	0								

WHEN I =2, W = 4

The weight corresponding to the value 2 is 4, i.e., $w_2 = 4$. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 4. We can put item of weight 4 in a knapsack as the profit corresponding to weight 4 is more than the item having weight 3, so we add 3 at M[2][4] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3				
3	0								
4	0								

WHEN I = 2, W = 5

The weight corresponding to the value 2 is 4, i.e., $w_2 = 4$. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 5. We can put item of weight 4 in a knapsack and the profit corresponding to weight is 3, so we add 3 at M[2][5] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0

1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3			
3	0								
4	0								

WHEN I = 2, W = 6

The weight corresponding to the value 2 is 4, i.e., $w_2 = 4$. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 6. We can put item of weight 4 in a knapsack and the profit corresponding to weight is 3, so we add 3 at M[2][6] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3		
3	0								
4	0								

WHEN I = 2, W = 7

The weight corresponding to the value 2 is 4, i.e., $w_2 = 4$. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 7. We can put item of weight 4 and 3 in a knapsack and the profits corresponding to weights are 2 and 3; therefore, the total profit is 5, so we add 5 at M[2][7] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	0	3	3	3	5	
3	0								
4	0								

WHEN I = 2, W = 8

The weight corresponding to the value 2 is 4, i.e., $w_2 = 4$. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 7. We can put item of weight 4 and 3 in a knapsack and the profits corresponding to weights are 2 and 3; therefore, the total profit is 5, so we add 5 at M[2][7] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0								
4	0								

NOW THE VALUE OF 'I' GETS INCREMENTED, AND BECOMES 3.

WHEN I = 3, W = 1

The weight corresponding to the value 3 is 5, i.e., $w_3 = 5$. Since we have three items in the set having weights 3, 4, and 5, and the weight of the knapsack is 1. We cannot put neither of the items in a knapsack, so we add 0 at M[3][1] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0							

4 0

WHEN I = 3, W = 2

The weight corresponding to the value 3 is 5, i.e., $w_3 = 5$. Since we have three items in the set having weight 3, 4, and 5, and the weight of the knapsack is 1. We cannot put neither of the items in a knapsack, so we add 0 at M[3][2] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0						
4	0								

WHEN I = 3, W = 3

The weight corresponding to the value 3 is 5, i.e., $w_3 = 5$. Since we have three items in the set of weight 3, 4, and 5 respectively and weight of the knapsack is 3. The item with a weight 3 can be put in the knapsack and the profit corresponding to the item is 2, so we add 2 at M[3][3] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	2					
4	0								

WHEN I = 3, W = 4

The weight corresponding to the value 3 is 5, i.e., $w_3 = 5$. Since we have three items in the set of weight 3, 4, and 5 respectively, and weight of the knapsack is 4. We can keep the item of either weight 3 or 4; the profit (3) corresponding to the weight 4 is more than the profit corresponding to the weight 3 so we add 3 at M[3][4] shown as below:

	0	1	2	3	4	5	6	7	8

0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3				
4	0								

WHEN I = 3, W = 5

The weight corresponding to the value 3 is 5, i.e., $w_3 = 5$. Since we have three items in the set of weight 3, 4, and 5 respectively, and weight of the knapsack is 5. We can keep the item of either weight 3, 4 or 5; the profit (3) corresponding to the weight 4 is more than the profits corresponding to the weight 3 and 5 so we add 3 at M[3][5] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3	3			
4	0								

WHEN I =3, W = 6

The weight corresponding to the value 3 is 5, i.e., $w_3 = 5$. Since we have three items in the set of weight 3, 4, and 5 respectively, and weight of the knapsack is 6. We can keep the item of either weight 3, 4 or 5; the profit (3) corresponding to the weight 4 is more than the profits corresponding to the weight 3 and 5 so we add 3 at M[3][6] shown as below:

0	1	2	3	4	5	6	7	8	
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5

3	0	0	0	1	3	3	3
4	0						

WHEN I =3, W = 7

The weight corresponding to the value 3 is 5, i.e., $w_3 = 5$. Since we have three items in the set of weight 3, 4, and 5 respectively, and weight of the knapsack is 7. In this case, we can keep both the items of weight 3 and 4, the sum of the profit would be equal to (2 + 3), i.e., 5, so we add 5 at M[3][7] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3	3	3	5	
4	0								

WHEN I = 3, W = 8

The weight corresponding to the value 3 is 5, i.e., $w_3 = 5$. Since we have three items in the set of weight 3, 4, and 5 respectively, and the weight of the knapsack is 8. In this case, we can keep both the items of weight 3 and 4, the sum of the profit would be equal to (2 + 3), i.e., 5, so we add 5 at M[3][8] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3	3	3	5	5
4	0								

NOW THE VALUE OF 'I' GETS INCREMENTED AND BECOMES

4.

WHEN I = 4, W = 1

The weight corresponding to the value 4 is 6, i.e., $w_4 = 6$. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 1. The weight of all the items is more than the weight of the knapsack, so we cannot add any item in the knapsack; Therefore, we add 0 at M[4][1] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3	3	3	5	5
4	0	0							

WHEN I = 4, W = 2

The weight corresponding to the value 4 is 6, i.e., $w_4 = 6$. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 2. The weight of all the items is more than the weight of the knapsack, so we cannot add any item in the knapsack; Therefore, we add 0 at M[4][2] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3	3	3	5	5
4	0	0	0						

WHEN I = 4, W = 3

The weight corresponding to the value 4 is 6, i.e., $w_4 = 6$. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 3. The item with a weight 3 can be put in the knapsack and the profit corresponding to the weight 4 is 2, so we will add 2 at M[4][3] shown as below:

	0	1	2	3	4	5	6	7	8

0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3	3	3	5	5
4	0	0	0	2					

WHEN I = 4, W = 4

The weight corresponding to the value 4 is 6, i.e., $w_4 = 6$. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 4. The item with a weight 4 can be put in the knapsack and the profit corresponding to the weight 4 is 3, so we will add 3 at M[4][4] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3	3	3	5	5
4	0	0	0	2	3				

WHEN I = 4, W = 5

The weight corresponding to the value 4 is 6, i.e., $w_4 = 6$. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 5. The item with a weight 4 can be put in the knapsack and the profit corresponding to the weight 4 is 3, so we will add 3 at M[4][5] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5

3	0	0	0	1	3	3	3	5	5
4	0	0	0	2	3	3			

WHEN I = 4, W = 6

The weight corresponding to the value 4 is 6, i.e., $w_4 = 6$. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 6. In this case, we can put the items in the knapsack either of weight 3, 4, 5 or 6 but the profit, i.e., 4 corresponding to the weight 6 is highest among all the items; therefore, we add 4 at M[4][6] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	1	3	3	3	5	5
4	0	0	0	2	3	3	4		

WHEN I = 4, W = 7

The weight corresponding to the value 4 is 6, i.e., $w_4 = 6$. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 7. Here, if we add two items of weights 3 and 4 then it will produce the maximum profit, i.e., (2 + 3) equals to 5, so we add 5 at M[4][7] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	2	3	3	3	5	5
4	0	0	0	2	3	3	4	5	

WHEN I = 4, W = 8

The weight corresponding to the value 4 is 6, i.e., $w_4 = 6$. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 8. Here, if we add two items of weights 3 and 4 then it will produce the maximum profit, i.e., (2 + 3) equals to 5, so we add 5 at

M[4][8] shown as below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	2	3	3	3	5	5
4	0	0	0	2	3	3	4	5	5

As we can observe in the above table that 5 is the maximum profit among all the entries. The pointer points to the last row and the last column having 5 value. Now we will compare 5 value with the previous row; if the previous row, i.e., i = 3 contains the same value 5 then the pointer will shift upwards. Since the previous row contains the value 5 so the pointer will be shifted upwards as shown in the below table:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	2	3	3	3	5	5
4	0	0	0	2	3	3	4	5	5

Again, we will compare the value 5 from the above row, i.e., i = 2. Since the above row contains the value 5 so the pointer will again be shifted upwards as shown in the below table:

0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	2	3	3	3	5	5
4	0	0	0	2	3	3	4	5	5

Again, we will compare the value 5 from the above row, i.e., i = 1. Since the above row does not contain the same value so we will consider the row i=1, and the weight corresponding to the row

is 4. Therefore, we have selected the weight 4 and we have rejected the weights 5 and 6 shown below:

$X = \{ 1, 0, 0\}$

The profit corresponding to the weight is 3. Therefore, the remaining profit is (5 - 3) equals to 2. Now we will compare this value 2 with the row i = 2. Since the row (i = 1) contains the value 2; therefore, the pointer shifted upwards shown below:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	2	3	3	3	5	5
4	0	0	0	2	3	3	4	5	5

Again we compare the value 2 with a above row, i.e., i = 1. Since the row i =0 does not contain the value 2, so row i = 1 will be selected and the weight corresponding to the i = 1 is 3 shown below:

$X = \{1, 1, 0, 0\}$

The profit corresponding to the weight is 2. Therefore, the remaining profit is 0. We compare 0 value with the above row. Since the above row contains a 0 value but the profit corresponding to this row is 0. In this problem, two weights are selected, i.e., 3 and 4 to maximize the profit.

ALGORITHM:

Algorithm KNAPSACK (V, W, M)

// Description: Solve binary knapsack problem using dynamic programming

// Input: Set of items X, set of weight W, profit of items V and knapsack capacity M

// Output: Array V, which holds the solution of problem

for i ← 1 to n **do**

V[i, 0] ← 0

END

for i ← 1 to M **do**

V[0, i] ← 0

END

for V[0, i] ← 0 **do**

for j ← 0 to M **do**

if w[i] ≤ j **then** V[i, j] ← max{V[i-1, j], v[i] + V[i – 1, j – w[i]]}

ELSE

V[i, j] ← V[i – 1, j] // w[i]>j

E

ND

END

END

Complexity analysis

Dynamic programming finds an optimal solution by constructing a table of size n ´ M, where n isa number of items and M is the capacity of the knapsack. This table can be filled up in O(n*m) time, same is the space complexity.

All Pairs Shortest Path (Floyd-Warshall) Algorithm

ALL PAIRS SHORTEST PATH ALGORITHM – INTRODUCTION

All Pairs Shortest Path Algorithm is also known as the Floyd-Warshall algorithm. And this is an optimization problem that can be solved using dynamic programming.
Let G = <V, E> be a directed graph, where V is a set of vertices and E is a set of edges with nonnegative length. Find the shortest path between each pair of nodes.
L = Matrix, which gives the length of each edge

L[i, j] = 0, if i == j // Distance of node from itself is zero

L[i, j] = ∞ , if i ≠ j and (i, j) ∉ E

L[i, j] = w (i, j), if i ≠ j and (i, j) ∈ E // w(i, j) is the weight of the edge (i, j)

PRINCIPLE OF OPTIMALITY :

If k is the node on the shortest path from i to j, then the path from i to k and k to j, must also be shortest.
In the following figure, the optimal path from i to j is either p or summation of p_1 and p_2.While considering k^{th} vertex as intermediate vertex, there are two possibilities :

- If k is not part of shortest path from i to j, we keep the distance D[i, j] as it is.

- If k is part of shortest path from i to j, update distance D[i, j] asD[i, k] + D[k, j].

Optimal sub structure of the problem is given as :

$D^k [i, j] = \min \{ D^{k-1} [i, j], D^{k-1} [i, k] + D^{k-1} [k, j] \}$
D^k = Distance matrix after k^{th} iteration

ALGORITHM FORALL PAIRS SHORTEST PATH

This approach is also known as the **Floyd-warshall** shortest path algorithm. The algorithm for all pair shortest path (APSP) problem is described below

```
Algorithm FLOYD_APSP ( L)

// L is the matrix of size n × n representing original graph
// D is the distance matrix

D ← L
for k ← 1 to n do
   for i ← 1 to n do

      for j ← 1 to n do

         D[i, j]^k ← min ( D[i, j]^k-1, D[i, k]^k-1 + D[k, j]^k-1 )
      end
   end
```

COMPLEXITY ANALYSIS OF ALL PAIRS SHORTEST PATH ALGORITHM

It is very simple to derive the complexity of all pairs' shortest path problem from the above algorithm. It uses three nested loops. The innermost loop has only one statement. The complexity of that statement is Q(1).

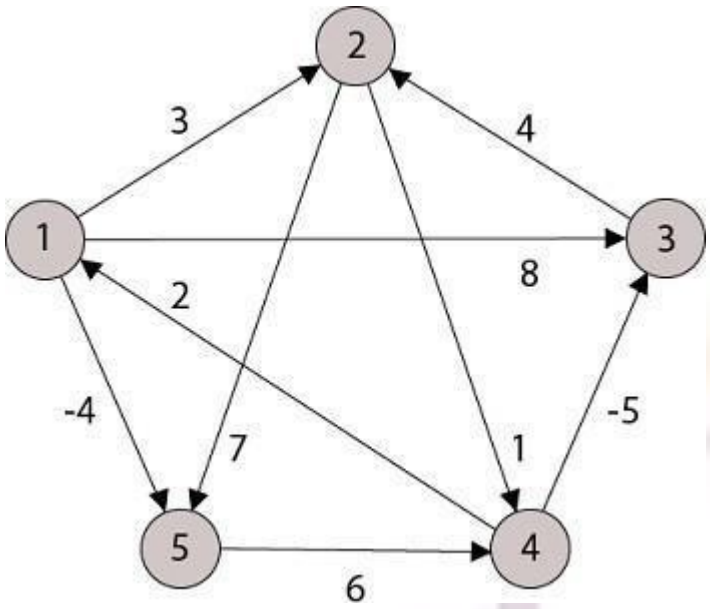
The running time of the algorithm is computed as :

$$T(n) = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \Theta(1) = \sum_{k=1}^{n} \sum_{i=1}^{n} n = \sum_{k=1}^{n} n^2 = O(n^3)$$

A recursive definition is given by

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k=0 \\ \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} & \text{if } k \geq 1) \end{cases}$$

Example: Apply Floyd-Warshall algorithm for constructing the shortest path. Show thatmatrices $D^{(k)}$ and $\pi^{(k)}$ computed by the Floyd-Warshall algorithm for the graph.



SOLUTION:

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} \text{ if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} \text{ if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Step (i) When k = 0

$$D^{(0)} = \begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & \infty \\ 2 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{matrix} \qquad \pi^{(0)} = \begin{matrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 3 & NIL \\ 4 & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{matrix}$$

(ii) When k =1

$d_{ij}^{(k)} = \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

$d_{14}^{(1)} = \min (d_{14}^{(0)}, d_{11}^{(0)} + d_{14}^{(0)})$

$d_{14}^{(1)} = \min (\infty , 0 + \infty) = \infty$

$d_{15}^{(1)} = \min (d_{15}^{(0)}, d_{11}^{(0)} + d_{15}^{(0)})$

$d_{15}^{(1)} = \min (-4, 0 + -4) = -4$

$d_{21}^{(1)} = \min (d_{21}^{(0)}, d_{21}^{(0)} + d_{11}^{(0)})$

$d_{21}^{(1)} = \min (\infty, \infty + 0) = \infty$

$d_{23}^{(1)} = \min (d_{23}^{(0)}, d_{21}^{(0)} + d_{13}^{(0)})$

$d_{23}^{(1)} = \min ((\infty, \infty + 8) = \infty$

$d_{31}^{(1)} = \min (d_{31}^{(0)}, d_{31}^{(0)} + d_{11}^{(0)})$

$d_{31}^{(1)} = \min (\infty, \infty + 0) = \infty$

$$D_{ij}^{(1)} = \begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & \infty \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix} \qquad \pi^{(1)} = \begin{matrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 3 & NIL \\ 4 & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{matrix}$$

Step (iii) When k = 2

$$d_{ij}^{(k)} = \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$d_{14}^{(2)} = \min (d_{14}^{(1)}, d_{12}^{(1)} + d_{24}^{(1)})$$

$$d_{14}^{(2)} = \min (\infty, 3 + 1) = 4$$

$$d_{21}^{(2)} = \min (d_{21}^{(1)}, d_{22}^{(1)} + d_{21}^{(1)})$$

$$d_{21}^{(2)} = \min (\infty , 0 + \infty) = \infty$$

$$d_{34}^{(2)} = \min (d_{34}^{(1)}, d_{32}^{(1)} + d_{24}^{(1)})$$

$$d_{34}^{(2)} = \min (-5, 4 + 1) = -5$$

$$d_{35}^{(2)} = \min (d_{35}^{(1)}, d_{32}^{(1)} + d_{25}^{(1)})$$

$$d_{35}^{(2)} = \min (\infty, 4 + 7) = 11$$

$$d_{43}^{(2)} = \min (d_{43}^{(1)}, d_{42}^{(1)} + d_{23}^{(1)})$$

$$d_{43}^{(2)} = \min (10, 5 + \infty) = 10$$

$$D_{ij}^{(2)} = \begin{matrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & 11 \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix} \qquad \pi^{(2)} = \begin{matrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 3 & 2 \\ 4 & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{matrix}$$

Step (iv) When k = 3

$$d_{ij}^{(k)} = \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$d_{14}^{(3)} = \min (d_{14}^{(2)}, d_{13}^{(2)} + d_{34}^{(2)})$$

$$d_{14}^{(3)} = \min (4, 8 + (-5)) = 3$$

$$D_{ij}^{(3)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & 11 \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix} \qquad \pi^{(3)} = \begin{matrix} NIL & 1 & 1 & 3 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 3 & 2 \\ 4 & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{matrix}$$

Step (v) When k = 4

$$d_{ij}^{(k)} = \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$d_{21}^{(4)} = \min (d_{21}^{(3)}, d_{24}^{(3)} + d_{41}^{(3)})$$

$$d_{21}^{(4)} = \min (\infty, 1 + 2) = 3$$

$$d_{23}^{(4)} = \min (d_{23}^{(3)}, d_{24}^{(3)} + d_{43}^{(3)})$$

$$d_{23}^{(4)} = \min (\infty, 1 + 10) = 11$$

$$d_{25}^{(4)} = \min (d_{25}^{(3)}, d_{24}^{(3)} + d_{45}^{(3)})$$

$$d_{25}^{(4)} = \min (7, 1 + (-2)) = -1$$

$$d_{31}^{(4)} = \min (d_{31}^{(3)}, d_{34}^{(3)} + d_{41}^{(3)})$$

$$d_{31}^{(4)} = \min (\infty, -5 + 2) = -3$$

$$d_{32}^{(4)} = \min (d_{32}^{(3)}, d_{34}^{(3)} + d_{42}^{(3)})$$

$$d_{32}^{(4)} = \min (4, -5 + 5) = 0$$

REDDY ENGINEERING COLLEGE

$$D_{ij}^{(4)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ 3 & 0 & 11 & 1 & -1 \\ -3 & 0 & 0 & -5 & -7 \\ 2 & 5 & 10 & 0 & -2 \\ 8 & 11 & 16 & 6 & 0 \end{matrix} \qquad \pi^{(4)} = \begin{matrix} NIL & 1 & 1 & 3 & 1 \\ 4 & NIL & 4 & 2 & 2 \\ 4 & 4 & NIL & 3 & 4 \\ 4 & 1 & 1 & NIL & 1 \\ 4 & 4 & 4 & 5 & NIL \end{matrix}$$

Step (vi) When k = 5

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

$$d_{25}^{(5)} = \min\left(d_{25}^{(4)}, d_{25}^{(4)} + d_{55}^{(3)}\right)$$

$$d_{25}^{(5)} = \min(-1, -1 + 0) = -1$$

$$d_{23}^{(5)} = \min\left(d_{23}^{(4)}, d_{25}^{(4)} + d_{53}^{(3)}\right)$$

$$d_{23}^{(5)} = \min(11, -1 + 16) = 11$$

$$d_{35}^{(5)} = \min\left(d_{35}^{(4)}, d_{35}^{(4)} + d_{55}^{(3)}\right)$$

$$d_{35}^{(5)} = \min(-7, -7 + 0) = -7$$

$$D_{ij}^{(5)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ 3 & 0 & 11 & 1 & -1 \\ -3 & 0 & 0 & -5 & -7 \\ 2 & 5 & 10 & 0 & -2 \\ 8 & 11 & 16 & 6 & 0 \end{matrix} \qquad \pi^{(5)} = \begin{matrix} NIL & 1 & 1 & 5 & 1 \\ 4 & NIL & 4 & 2 & 4 \\ 4 & 4 & NIL & 3 & 4 \\ 4 & 1 & 1 & NIL & 1 \\ 4 & 4 & 4 & 5 & NIL \end{matrix}$$

Traveling Salesman Problem – Solve it using Dynamic Programming

BY CODECRUCKS · 09/12/2021

Traveling salesman problem (TSP) is the well studied and well-explored problem of computer science. Due to its application in diverse fields, TSP has been one of the most interesting problems for researchers and mathematicians.

TRAVELING SALESMAN PROBLEM – DESCRIPTION

- Traveling salesman problem is stated as, "Given a set of *n* cities and distance between eachpair of cities, find the minimum length path such that it covers each city exactly once and terminates the tour at starting city."
- It is not difficult to show that this problem is NP complete problem. There exists *n!*

paths, asearch of the optimal path becomes very slow when *n* is considerably large.

- Each edge (u, v) in TSP graph is assigned some non-negative weight, which representsthe distance
- between city u and v. This problem can be solved by finding the Hamiltonian cycle of thegraph.
- The distance between cities is best described by the weighted graph, where edge (u, v) indicates the path from city u to v and w(u, v) represents the distance between cities u andv.
- Let us formulate the solution of TSP using dynamic programming.

- From following figure, d[i, j] = min(d[i, j], d[i, k] + d[k, j])
- Dynamic programming always selects the path which is minimum.

ALGORITHM FOR TRAVELING SALESMAN PROBLEM:

Step 1:

Let d[i, j] indicates the distance between cities i and j. Function C[x, V – { x }]is the cost of thepath starting from city x. V is the set of cities/vertices in given graph. The aim of TSP is to minimize the cost function.

Step 2:

Assume that graph contains n vertices V1, V2, ..., Vn. TSP finds a path covering all verticesexactly once, and the same time it tries to minimize the overall traveling distance.

Step 3:

Mathematical formula to find minimum distance is stated below:C(i, V) = min { d[i, j] + C(j, V – { j }) }, j ∈ V and i ∉ V.

TSP problem possesses the principle of optimality, i.e. for d[V1, Vn] to be minimum, any

Complexity Analysis of Traveling salesman problem

Dynamic programming creates $n.2^n$ subproblems for n cities. Each sub-problem can be solved in linear time. Thus the time complexity of TSP using dynamic programming would be $O(n^2 2^n)$. It is much less than n! but still, it is an exponent. Space complexity is also exponential.

EXAMPLE

Problem: Solve the traveling salesman problem with the associated cost adjacency matrixusing dynamic programming.

–	24	11	10	9
8	–	2	5	11

26	12	–	8	7
11	23	24	–	6
5	4	8	11	–

SOLUTION:

Let us start our tour from city 1.

Step 1: Initially, we will find the distance between city 1 and city {2, 3, 4, 5} without visiting any intermediate city.
Cost(x, y, z) represents the distance from x to z and y as an intermediate city.

Cost(2, Φ, 1) = d[2, 1] = 24

Cost(3, Φ, 1) = d[3, 1] = 11

Cost(4, Φ , 1) = d[4, 1] = 10

Cost(5, Φ , 1) = d[5, 1] = 9

Step 2: In this step, we will find the minimum distance by visiting 1 city as intermediate city.

Cost{2, {3}, 1} = d[2, 3] + Cost(3, f, 1)

= 2 + 11 = 13

Cost{2, {4}, 1} = d[2, 4] + Cost(4, f, 1)

= 5 + 10 = 15

Cost{2, {5}, 1} = d[2, 5] + Cost(5, f, 1)

= 11 + 9 = 20

Cost{3, {2}, 1} = d[3, 2] + Cost(2, f, 1)

= 12 + 24 = 36

Cost{3, {4}, 1} = d[3, 4] + Cost(4, f, 1)

= 8 + 10 = 18

Cost{3, {5}, 1} = d[3, 5] + Cost(5, f, 1)

= 7 + 9 = 16

Cost{4, {2}, 1} = d[4, 2] + Cost(2, f, 1)

= 23 + 24 = 47

Cost{4, {3}, 1} = d[4, 3] + Cost(3, f, 1)

= 24 + 11 = 35

Cost{4, {5}, 1} = d[4, 5] + Cost(5, f, 1)

= 6 + 9 = 15

Cost{5, {2}, 1} = d[5, 2] + Cost(2, f, 1)

= 4 + 24 = 28

Cost{5, {3}, 1} = d[5, 3] + Cost(3, f, 1)

= 8 + 11 = 19

Cost{5, {4}, 1} = d[5, 4] + Cost(4, f, 1)

= 11 + 10 = 21

Step 3: In this step, we will find the minimum distance by visiting 2 cities as intermediate city.Cost(2, {3, 4}, 1) = min { d[2, 3] + Cost(3, {4}, 1), d[2, 4] + Cost(4, {3}, 1)]}

= min { [2 + 18], [5 + 35] }

= min{20, 40} = 20

Cost(2, {4, 5}, 1) = min { d[2, 4] + Cost(4, {5}, 1), d[2, 5] + Cost(5, {4}, 1)]}

= min { [5 + 15], [11 + 21] }

= min{20, 32} = 20

Cost(2, {3, 5}, 1) = min { d[2, 3] + Cost(3, {4}, 1), d[2, 4] + Cost(4, {3}, 1)]}

= min { [2 + 18], [5 + 35] }

= min{20, 40} = 20

Cost(3, {2, 4}, 1) = min { d[3, 2] + Cost(2, {4}, 1), d[3, 4] + Cost(4, {2}, 1)]}

= min { [12 + 15], [8 + 47] }

= min{27, 55} = 27

Cost(3, {4, 5}, 1) = min { d[3, 4] + Cost(4, {5}, 1), d[3, 5] + Cost(5, {4}, 1)]}
= min { [8 + 15], [7 + 21] }

= min{23, 28} = 23

Cost(3, {2, 5}, 1) = min { d[3, 2] + Cost(2, {5}, 1), d[3, 5] + Cost(5, {2}, 1)]}

= min { [12 + 20], [7 + 28] }

= min{32, 35} = 32

Cost(4, {2, 3}, 1) = min{ d[4, 2] + Cost(2, {3}, 1), d[4, 3] + Cost(3, {2}, 1)]}

= min { [23 + 13], [24 + 36] }

= min{36, 60} = 36

Cost(4, {3, 5}, 1) = min{ d[4, 3] + Cost(3, {5}, 1), d[4, 5] + Cost(5, {3}, 1)]}

= min { [24 + 16], [6 + 19] }

= min{40, 25} = 25

Cost(4, {2, 5}, 1) = min{ d[4, 2] + Cost(2, {5}, 1), d[4, 5] + Cost(5, {2}, 1)]}

= min { [23 + 20], [6 + 28] }

= min{43, 34} = 34

Cost(5, {2, 3}, 1) = min{ d[5, 2] + Cost(2, {3}, 1), d[5, 3] + Cost(3, {2}, 1)]}

= min { [4 + 13], [8 + 36] }

= min{17, 44} = 17

Cost(5, {3, 4}, 1) = min{ d[5, 3] + Cost(3, {4}, 1), d[5, 4] + Cost(4, {3}, 1)]}

= min { [8 + 18], [11 + 35] }

= min{26, 46} = 26

Cost(5, {2, 4}, 1) = min{ d[5, 2] + Cost(2, {4}, 1), d[5, 4] + Cost(4, {2}, 1)]}

= min { [4 + 15], [11 + 47] }

= min{19, 58} = 19

Step 4 : In this step, we will find the minimum distance by visiting 3 cities as intermediatecity.

Cost(2, {3, 4, 5}, 1) = min { d[2, 3] + Cost(3, {4, 5}, 1), d[2, 4] + Cost(4, {3, 5}, 1), d[2, 5] + Cost(5, {3, 4}, 1)}

= min { 2 + 23, 5 + 25, 11 + 36}
= min{25, 30, 47} = 25

Cost(3, {2, 4, 5}, 1) = min { d[3, 2] + Cost(2, {4, 5}, 1), d[3, 4] + Cost(4, {2, 5}, 1), d[3, 5] + Cost(5, {2, 4}, 1)}

= min { 12 + 20, 8 + 34, 7 + 19}

= min{32, 42, 26} = 26

Cost(4, {2, 3, 5}, 1) = min { d[4, 2] + Cost(2, {3, 5}, 1), d[4, 3] + Cost(3, {2, 5}, 1), d[4, 5] + Cost(5, {2, 3}, 1)}

= min {23 + 30, 24 + 32, 6 + 17}

= min{53, 56, 23} = 23

Cost(5, {2, 3, 4}, 1) = min { d[5, 2] + Cost(2, {3, 4}, 1), d[5, 3] + Cost(3, {2, 4}, 1), d[5, 4] + Cost(4, {2, 3}, 1)}

= min {4 + 30, 8 + 27, 11 + 36}

= min{34, 35, 47} = 34

Step 5 : In this step, we will find the minimum distance by visiting 4 cities as an intermediate city.
Cost(1, {2, 3, 4, 5}, 1) = min { d[1, 2] + Cost(2, {3, 4, 5}, 1), d[1, 3] + Cost(3, {2, 4, 5}, 1), d[1, 4] + Cost(4, {2, 3, 5}, 1) , d[1, 5] + Cost(5, {2, 3, 4}, 1)}

= min { 24 + 25, 11 + 26, 10 + 23, 9 + 34 }

= min{49, 37, 33, 43} = 33

Thus, minimum length tour would be of 33.

TRACE THE PATH:

- Let us find the path that gives the distance of 33.
- Cost(1, {2, 3, 4, 5}, 1) is minimum due to d[1, 4], so move from 1 to 4. Path = {1, 4}.
  - Cost(4, {2, 3, 5}, 1) is minimum due to d[4, 5], so move from 4 to 5. Path = {1, 4, 5}.
  - Cost(5, {2, 3}, 1) is minimum due to d[5, 2], so move from 5 to 2. Path = {1, 4, 5, 2}.
  - Cost(2, {3}, 1) is minimum due to d[2, 3], so move from 2 to 3. Path = {1, 4, 5, 2, 3}. All cities are visited so come back to 1. Hence the optimum tour would be 1 – 4 – 5 – 2 – 3 – 1.

RELIABILITY DESIGN:

Reliability means the ability of an apparatus, machine, or system to consistently perform its

intendedor required function or mission, on demand and without degradation or failure.

Reliability design using dynamic programming is used to solve a problem with a multiplicative optimization function. The problem is to design a system which is composed of several devices connected in series (below Fig-3.3(b)). Let *r;* be the reliability of device *D;* (i.e. *r;* is the probability that device *i* will function properly). Then, the reliability of the entire system is πr_i. Even if the individual devices are very reliable (the r_i'*s* are very close to one), the reliability of the system may not be very good.

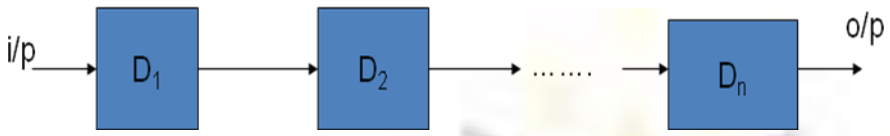


Fig-3.3(a) Devices connected in series

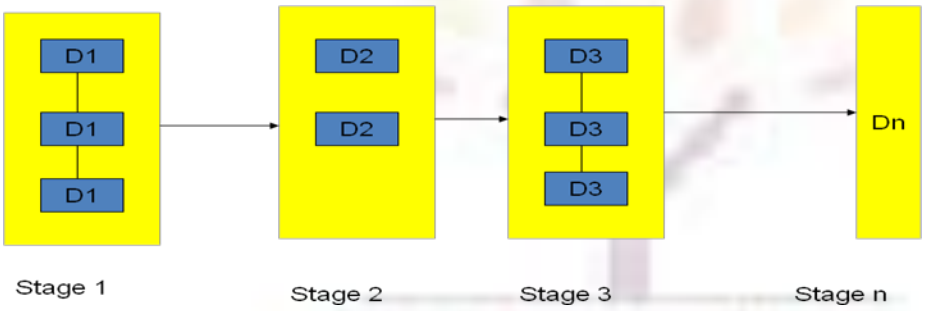


Fig-3.3(b) Multiple Devices Connected in Parallel in each stage

Multiple copies of the same device type are connected in parallel (Fig-3.3(b)) through the use of switching circuits. The switching circuits determine which devices in any given group are functioning properly. They then make use of one such device at each stage.

If stage i contains m copies of device D then the probability that all m have a malfunction is $(1 - r_i)^{m_i}$. Hence the reliability of stage i becomes $1 - (1 - r_i)^{m_i}$. Thus, if $r_i = 0.99$ and $m_i = 2$ the stage reliability becomes 0.9999. In any practical situation, the stage reliability will be a little less than $1 - (1 - r_i)^{m_i}$ because the switching circuits themselves are not fully reliable. Also,failures of copies of the same device may not be fully independent (e.g. if failure is due to design defect). Let us assume that the reliability of stage *i* is actually given by a function $\Phi_i(m_i)$, $1<=i<=n$. (It is quite conceivable that $\Phi_i(m_i)$ may decrease after a certain value of *m* ;). The reliability of thesystem of stages is $\prod_{1<=i<=n} \Phi_i(m_i)$.

Our problem is to use device duplication to maximize reliability. This maximization is to be carried out under a cost constraint.

Let c_i be the cost of each unit of device *i* and let *c* be the maximum allowable cost of the systembeing

designed.

We wish to solve the following maximization problem:
maximize $\prod_{1<=i<=n} \Phi_i(m_i)$

subject to $\sum_{1<=i<=n} c_i m_i <= cmi >= 1$ and integer i, $1<=i<=n$

A dynamic programming solution may be obtained in a manner similar to that used for the knapsackproblem. Since, we may assume each $c_i > 0$, each m_i must be in the range $1<=m_i<=u_i$ where

$= \lfloor (C + CI - \sum_{1 \text{ TO } N} CJ) / CI \rfloor$

The upper bound u_i follows from the observation that $mj>=1$. An optimal solution $m_1, m_2, \ldots, m_n$ is the result of a sequence of decisions, one decision for each m_i.
Letfi(x) represent the maximum value of $\Phi(mj)$, $1<=j<=i$ subject to the constraints $\sum_{1<=j<=i} c_j m_j <= x$ and $1<=mj<=uj$, $1<=j<=i$. Then, the value of an optimal solution is $f_n(c)$. Thelast decision made requires one to choose m_n from one of $\{1, 2, 3, \ldots, u_n\}$. Once a value for m_n has been chosen, the remaining decisions must be such as to use the remaining funds $c - c_n m_n$ in anoptimal way. The principal of optimality holds and

$$f_n(c) = \max_{1<=mn<=un} \{ \Phi n(m_n) f_{n-1}(c - c_n m_n) \}$$

For any $fi(x)$,$i>=1$ this equation generalizes to

$$F_I(X) = MAX_{1<=MI<=UI} \{ \Phi_I(MI)$$

$$FI\text{-}1(C - CIMI) \}$$

PROBLEMS BASED ON RELIABILITY

DESIGN:

Q.1 Design a three stage system with device types D1,D2,D3. The costs are Rs. 30, Rs. 15 and Rs.20 respectively. The cost of the system is to be no more than Rs. 105. The reliability of each device type is 0.9,0.8 and 0.5 respectively.

SOLUTION:

We will first compute u_1, u_2, u_3 using followingformula.$u_i = (C + C_i - \text{sigma } C_j)/ C_i$
For computing u_i

$u_1 = 2$(approx value)

For computing u_2

$u_2 = 3$(approx value)

For computing u_3

$u_3 = 3$

Hence (u_1 ,u_2 ,u_3)

Computing subsequences-

$S^0 = (1,0)$

Let Si consist of tuples of the form (f, x)

$=(r,c)S^0 = \{(1,0)\}$

For device D_1

for 1 D_1

$r_1=0.9, c_1=30$

1 $S^1 = \{(0.9, 30)\}$

For device D_1 for 2 D_1

$m_1=2$(2 D_1 device in parallel) Reliability of stage $1 = 1-(1- r_1)^2$

Reliability of stage $1 = 1-(1- 0.9)^2 =$ 0.99Cost =30*2 = 60

2 $S^1 = \{(0.99, 60)\}$

$S^1 = \{(0.9, 30), (0.99, 60)\}$

$S^1 = \{(0.9, 30), (0.99, 60)\}$

For one Device D2:-

$S_1 = \{(0.72, 45), (0.792, 75)\}$

For two Device D2:-

2 $S^2 = \{(0.864, 60), (0.9504, 90)\}$

For three Device D2:-

3 $S^2 = \{(0.8928, 75), (0.98208, 105)\}$

$S^2 = \{(0.72, 45), (0.792, 75), (0.864, 60),(0.9504, 90), (0.8928, 75), (0.98208, 105)\}$

(0.792, 75), (0.9504, 90) is eleminated due to purging or dominance rule and (0.98208, 105) iseleminated due to access cost 105.

After this we got

$S^2 = \{(0.72, 45), (0.864, 60), (0.8928, 75)\}$

For one Device D3:-

$S_1 = \{(0.36, 65), (0.432, 80), (0.4464, 95)\}$

For Two Device D3:-

$S_2 = \{(0.54, 85), (0.648, 100)\}$

FOR THREE DEVICE D3:-

$S^3 = \{(0.63, 105)\}_3$

Now we are going to find S^3

S^3 = { (0.36, 65), (0.432, 80), (0.4464, 95), (0.54, 85), (0.648, 100), (0.63, 105) }
Due to purging rule after elimination we get
S^3 = { (0.36, 65), (0.432, 80), (0.54, 85), (0.648, 100),}
Now
The best design has a
reliability of0.648 and a cost
of 100.Tracing back through
S^i's
WE DETERMINE THAT $M1 = 1$, $M2 = 2$, $M3 = 2$

UNIT-IV

Greedy method: General method, applications-Job sequencing with deadlines, knapsack problem, Minimum cost spanning trees, Single source shortest path problem.

Greedy Method:

The greedy method is perhaps (maybe or possible) the most straight forward designtechnique, used to determine a feasiblesolution that may or may not be optimal.

Feasible solution:- Most problems have n inputs and its solution contains a subset of inputsthat satisfies a given constraint(condition). Any subset that satisfies the constraint is called feasible solution.

Optimal solution: To find a feasible solution that either maximizes or minimizes a givenobjective function. A feasible solution that does thisis called optimal solution.

Application of Greedy

- **Method:** Job
- sequencing with
- deadline0/1
- knapsack problem
- Minimum cost spanning trees
- Single source shortest path problem.

JOB SEQUENCE WITH DEADLINE:

There is set of n-jobs. For any job i, is a integer deadling di≥0 and profitPi>0, the profit Pi isearned iff the job completed by its deadline.

To complete a job one had to process the job on a machine for one unitof time. Only onemachine is available for processing jobs.

A feasible solution for this problem is a subset J of jobs such that eachjob in this subset canbe completed by its deadline.

The value of a feasible solution J is the sum of the profits of the jobs in J, i.e., $\sum i \in J P_i$

An optimal solution is a feasible solution with maximum value.

The problem involves identification of a subset of jobs which can be completed by its deadline. Therefore the problem suites the subset methodology and can be solved by the greedy method.

Ex: - Obtain the optimal sequence for the following jobs.

j_1 j_2 j_3 j_4

(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)

(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)

n =4

Feasible solution	Processing sequence	Value
j_1 j_2 (1, 2)	(2,1)	100+10=110
(1,3)	(1,3) or (3,1)	100+15=115
(1,4)	(4,1)	100+27=127
(2,3)	(2,3)	10+15=25
(3,4)	(4,3)	15+27=42
(1)	(1)	100
(2)	(2)	10
(3)	(3)	15
(4)	(4)	27

In the example solution '3' is the optimal. In this solution only jobs 1&4 are processed and the value is 127. These jobs must be processed in the order j4 followed by j1. the process of job 4 begins at time 0 and ends at time 1. And the processing of job 1 begins at time 1 and ends at time2. Therefore both the jobs are completed within their deadlines. The optimization measure for determining the next job to be selected in to the solution is according to the profit. The next job to include is that which increases ∑pi the most, subject to the constraint that the resulting "j"is the feasible solution. Therefore the greedy strategy is to consider the jobsin decreasing order of profits.

The greedy algorithm is used to obtain an optimal solution.

We must formulate an optimization measure to determine how the next job is chosen.

```
algorithm js(d, j, n)
//d→ dead line, j→subset of jobs ,n→ total number of jobs
// d[i]≥1 1 ≤ i ≤ n are the dead lines,
// the jobs are ordered such that p[1]≥p[2]≥---≥p[n]
//j[i] is the ith job in the optimal solution 1 ≤ i ≤ k, k→ subset range
{
d[0]=j[
0]=0;
j[1]=1;
k=1;
for i=2 ton
do{r=k;
while((d[j[r]]>d[i]) and
[d[j[r]]≠r)) dor=r-1; if((d[j[r]]≤d[i])
and (d[i]> r)) then
{
for q:=k to (r+1) setp-1 do j[q+1]=
j[q];j[r+1]=i;
k=k+1;
}
}
```

NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE

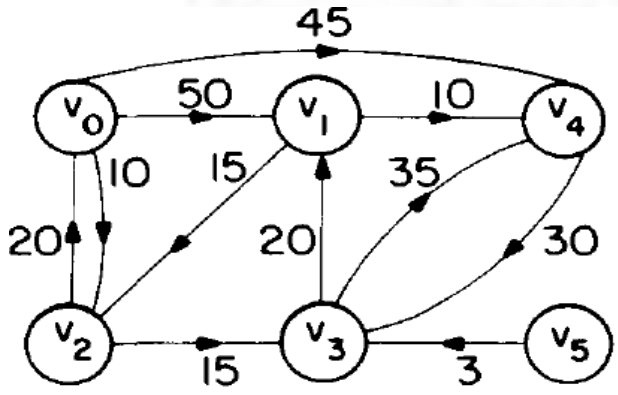
SINGLE SOURCE SHORTEST PATHS:

- Graphs can be used to represent the highway structure of a state or country with vertices representing cities and edges representing sections of highway.
- The edges have assigned weights which may be either the distance between the 2 cities connected by the edge or the average time to drive along that section ofhighway.
- For example if A motorist wishing to drive from city A to B thenwe must answer the following questions
  - Is there a path from A to B
  - If there is more than one path from A to B which is the shortest path
- The length of a path is defined to be the sum of the weights of the edges on that path.

Given a directed graph G(V,E) with weight edge w(u,v). e have to find a

shortest path fromsource vertex S∈v to every other vertex v1∈ v-s.

- To find SSSP for directed graphs G(V,E) there are two different algorithms.
  - Bellman-Ford Algorithm
  - Dijkstra's algorithm
- Bellman-Ford Algorithm:- allow –ve weight edges in input graph. Thisalgorithm either finds a shortest path form source vertex **S∈V** to other vertex**v∈V** or detect a –ve weight cycles in G, hence no solution. If there is no negative weight cycles are reachable form source vertex **S∈V** to every other vertex **v∈V**
- Dijkstra's algorithm:- allows only +ve weight edges in the inputgraph and finds ashortest path from source vertex **S∈V** to every other vertex **v∈V**.



	Path	Length
1)	$v_0\ v_2$	10
2)	$v_0\ v_2\ v_3$	25
3)	$v_0\ v_2\ v_3\ v_1$	45
4)	$v_0\ v_4$	45

Graph and shortest paths from v_0 to all destinations

- Consider the above directed graph, if node 1 is the source vertex,then shortest pathfrom 1 to 2 is 1,4,5,2. The length is 10+15+20=45.
- To formulate a greedy based algorithm to generate the shortest paths, we mustconceive of a multistage solution to the problem andalso of an optimization measure.
- This is possible by building the shortest paths one by one.
- As an optimization measure we can use the sum of the lengthsof all paths so fargenerated.
- If we have already constructed 'i' shortest paths, then using this optimization measure,the next path to be constructed should be thenext shortest minimum length path.
- The greedy way to generate the shortest paths from Vo to the remaining vertices is togenerate these paths in non-decreasingorder of path length.
- For this 1st, a shortest path of the nearest vertex is generated. Then a shortest path tothe 2nd nearest vertex is generated and so on.

Algorithm for finding Shortest Path

```
Algorithm ShortestPath(v, cost, dist, n)
//dist[j], 1≤j≤n, is set to the length of the shortest path from vertex v to vertex j in graph g
with n-vertices.
// dist[v] is zero
{
for i=1 to n do{
s[i]=false;
dist[i]=cost[v,i];
}
s[v]=true;
dist[v]:=0.0; // put v in s
for num=2 to n do{
// determine n-1 paths from v
choose u form among those vertices not in s such that dist[u] is minimum.
s[u]=true; // put u in s
for (each w adjacent to u with s[w]=false) do
if(dist[w]>(dist[u]+cost[u, w])) then
dist[w]=dist[u]+cost[u, w];
}
}
```

MINIMUM COST SPANNING TREE:

SPANNING TREE: - A Sub graph 'n' of o graph 'G' is called as a spanning tree if

(i) It includes all the vertices of 'G'

(ii) It is a tree

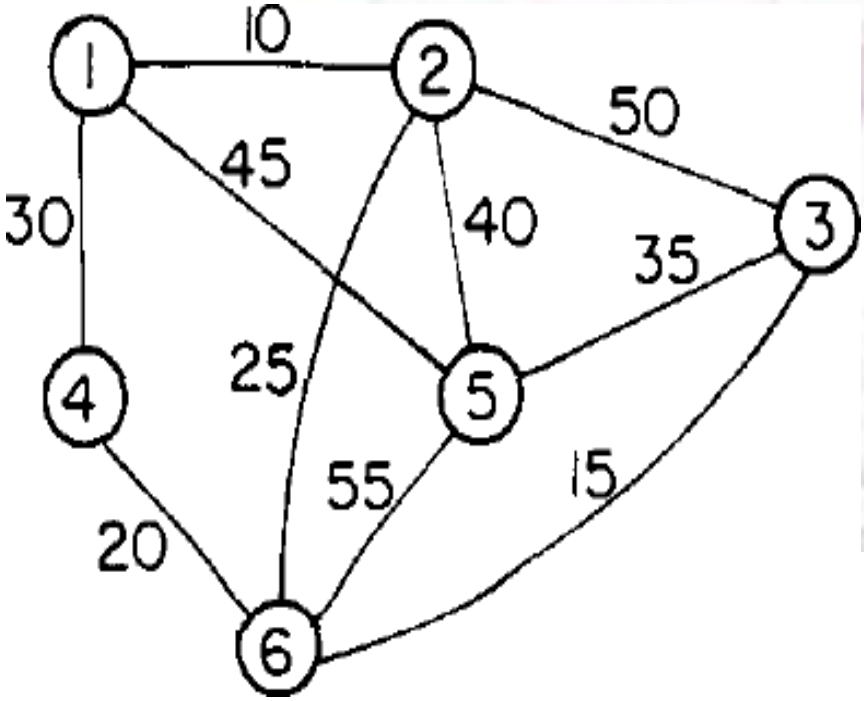
Minimum cost spanning tree: For a given graph 'G' there can be more than one spanning tree. If weights are assigned to the edges of 'G' then the spanning tree which has the minimum cost of edges is called as minimal spanning tree.

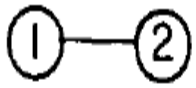
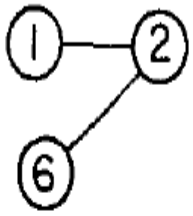
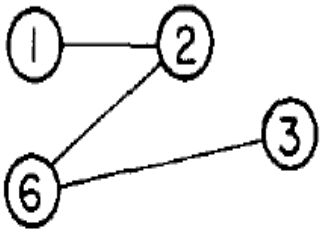
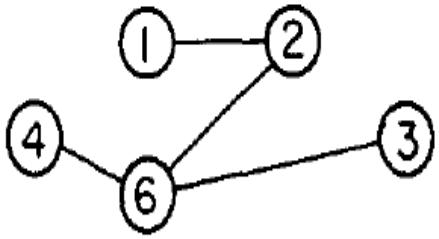
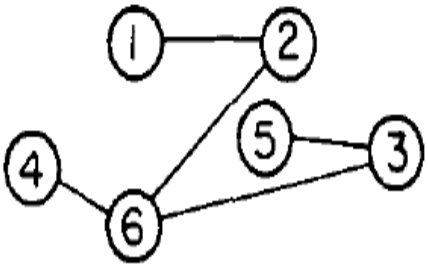
The greedy method suggests that a minimum cost spanning tree can be obtained by contactingthe tree edge by edge. The next edge to be included in the tree is the edge that results in a minimum increase in the some of the costs of the edges included so far.

There are two basic algorithms for finding minimum-cost spanning trees,and both are greedyalgorithms

- Prim's Algorithm
- Kruskal's Algorithm

Prim's Algorithm: Start with any *one node* in the spanning tree, and repeatedly add thecheapest edge, and the node it leads to, for which the node is not already in the spanning tree.



Edge	Cost	Spanning tree
(1,2)	10	
(2,6)	25	
(3,6)	15	
(6,4)	20	
(1,4)	reject	
(3,5)	35	

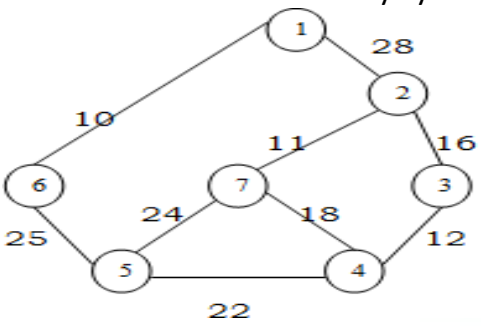
Stages in Prim's Algorithm

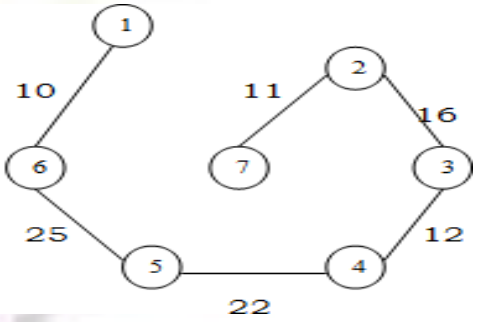
PRIM'S ALGORITHM: -

i) Select an edge with minimum cost and include in to the spanningtree.

ii) Among all the edges which are adjacent with theselected edge, select the onewith minimum cost.

iii) Repeat step 2 until 'n' vertices and (n-1) edges are been included. And the subgraph obtained does not contain any cycles.





NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE

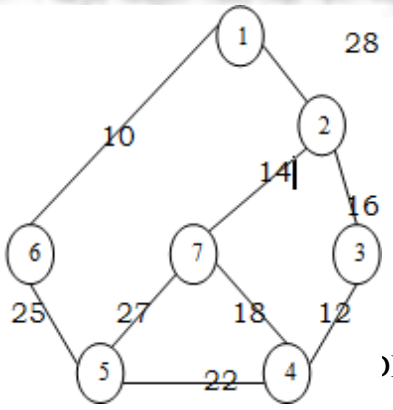
Prim's minimum spanning tree algorithm

```
Algorithm Prim (E, cost, n,t)
// E is the set of edges in G. Cost (1:n, 1:n) is the
// Cost adjacency matrix of an n vertex graph such that
// Cost (i,j) is either a positive real no. or ∞ if no edge (i,j) exists.
//A minimum spanning tree is computed and
//Stored in the array T(1:n-1, 2).
//(t (i, 1), + t(i,2)) is an edge in the minimum cost spanning tree. The final cost is
returned
        {
        Let (k, l) be an edge with min
        cost in EMin cost: = Cost (x,l);
        T(1,1):= k; + (1,2):= l;
for i:= 1 to n do//initialize near
        if (cost (i,l)<cost (i,k) then n east
        (i): l;else near (i): = k;
        near (k): = near (l):
        = 0;for i: = 2 to n-1
        do
{//find n-2 additional edges for t
let j be an index such that near (i)     0 & cost (j, near (i)) is
minimum;t (i,1): = j + (i,2): = near (j);
min cost: = Min cost + cost (j,
near (j)); near (j): = 0;
for k:=1 to n do // update near ()
if ((near (k)     0) and (cost {k, near (k)) >
cost (k,j)))then near Z(k): = ji
}
return mincost;
}
```

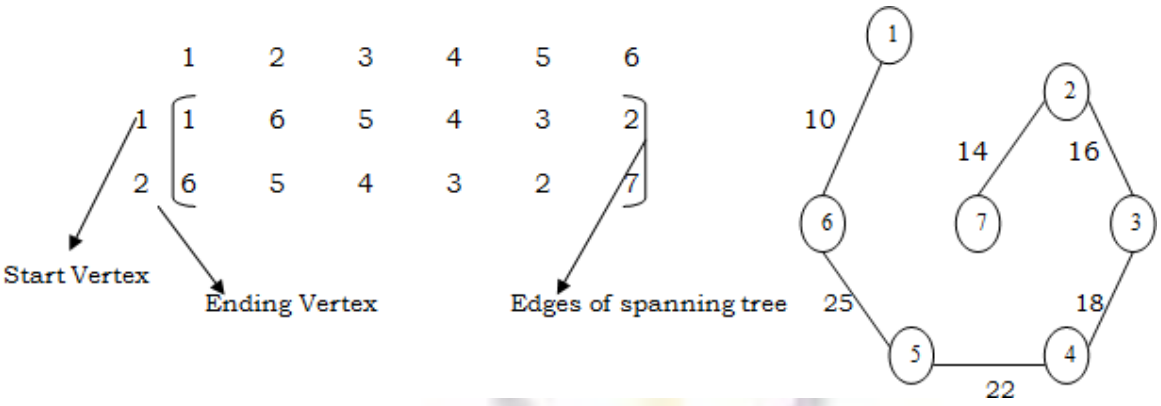
The algorithm takes four arguments E: set of edges, cost is nxn adjacency matrix cost of (i,j)= +ve integer, if an edge exists between i&j otherwise infinity. 'n' is no/: of vertices. 't' is a (n- 1):2matrix which consists of the edges of spanning tree.

E = { (1,2), (1,6), (2,3), (3,4), (4,5), (4,7), (5,6), (5,7), (2,7) }

Cost	1	2	3	4	5	6	7
1	α	28	α	α	α	10	α
2	28	α	16	α	α	α	14
3	α	10	α	12	α	α	α
4	α	α	12	α	22	α	18
5	α	α	α	22	α	25	24
6	10	α	α	α	25	α	α
7	α	14	α	18	24	α	α



n = {1,2,3,4,5,6,7)



i) The algorithm will start with a tree that includes only minimum cost edge of G. Then edges are added to this tree one by one.

ii) The next edge (i,j) to be added is such that i is a vertex which is already included in the treed and j is a vertex not yet included in the tree and cost of i,j is minimum among all edges adjacent to 'i'.

iii) With each vertex 'j' next yet included in the tree, we assign a value near 'j'. The value near 'j' represents a vertex in the tree such that cost (j, near (j)) is minimum among all choices for near (j)

iv) We define near (j):= 0 for all the vertices 'j' that are already in the tree.

v) The next edge to include is defined by the vertex 'j' such that (near (j)) 0 and cost of (j, near (j)) is minimum.

Analysis: -

The time required by the prince algorithm is directly proportional to the no/: of vertices. If agraph 'G' has 'n' vertices then the time required by prim's algorithm is $\mathbf{0(n^2)}$

KRUSKAL'S ALGORITHM:

Start with *no* nodes or edges in the spanning tree, and repeatedly add the cheapest edge that does not create a cycle.
In Kruskals algorithm for determining the spanning tree we arrangethe edges in the increasing order of cost.

i) All the edges are considered one by one in that order and deleted from the graph and are included in to the spanning tree.

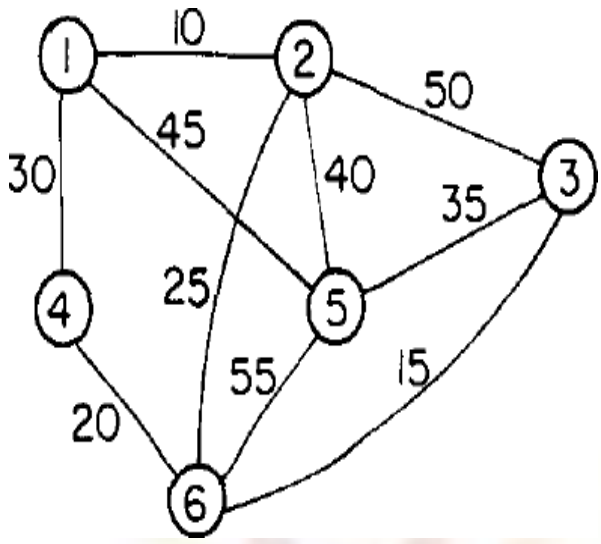
ii) At every stage an edge is included; the sub-graph at a stage need not be a tree. Infect it is a forest.

iii) At the end if we include 'n' vertices and n-1 edges without forming cycles then we get a single connected component without any cycles i.e. a tree with minimum cost.

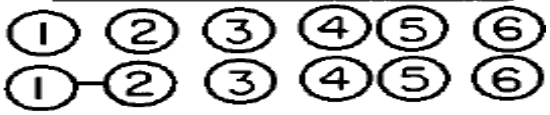
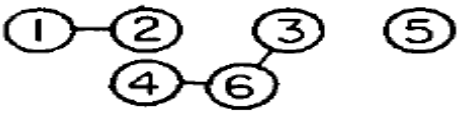
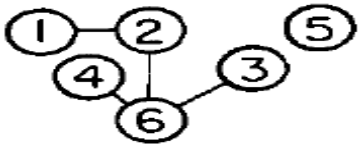
At every stage, as we include an edge in to the spanning tree, we get disconnected trees represented by various sets. While including an edge in to the spanning tree we need to check it does not form cycle. Inclusion of an edge (i,j) will form a cycle if i,j both are in same set. Otherwise the edge can be included into the spanning tree.

Kruskal minimum spanning tree algorithm

```
Algorithm Kruskal (E, cost, n,t)
//E is the set of edges in G. 'G' has 'n' vertices
//Cost {u,v} is the cost of edge (u,v) t is the set
//of edges in the minimum cost spanning tree
//The final cost is returned
{ construct a heap out of the edge costs using heapify;
        for i:= 1 to n do parent (i):= -1 // place in different sets
//each vertex is in different set      {1} {1}
        {3}i: = 0; min cost: = 0.0;
        While (i<n-1) and (heap not empty))do
{
Delete a minimum cost edge (u,v) from the heaps; and reheapify
using adjust;j:= find (u); k:=find (v);
if (j  k) then
{  i: = 1+1;
   + (i,1)=u; + (i, 2)=v;
    mincost: =
    mincost+cost(u,v);
    Union (j,k);
    }
}
if (i n-1) then write ("No
   spanning tree");else return
   mincost;
}
```



Consider the above graph of , Using Kruskal's method the edges of this graph are considered for inclusion in the minimum cost spanning tree in the order (1, 2), (3, 6), (4, 6), (2, 6), (1, 4),
(3, 5), (2, 5), (1, 5), (2, 3), and (5, 6). This corresponds to the cost sequence 10, 15, 20, 25, 30, 35, 40, 45, 50, 55. The first four edges are included in T. The next edge to be consideredis (I, 4). This edge connects two vertices already connected in T and so it is rejected. Next, the edge(3, 5) is selected and that completes the spanning tree.

Edge	Cast	Spanning Forest
		
(1,2)	10	
(3,6)	15	
(4,6)	20	
(2,6)	25	
(1,4)	30	(reject)
(3,5)	35	

Analysis: - If the no/: of edges in the graph is given by /E/then the time for Kruskalsalgorithm is given by 0 (|E| log |E|).

0/1 KNAPSACK PROBLEM:

Let there be n items, z_1 to z_n where z_i has a value v_i and weight w_i. The maximum weight that we can carry in the bag is *W*. It is common to assume that all values and weights are nonnegative. To simplify the representation, we also assume that the items are listed in increasing order of weight.

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i \quad \text{subject to} \sum_{i=1}^{n} w_i x_i \leqslant W, \qquad x_i \in \{0,1\}$$

Maximize the sum of the values of the items in the knapsack so that the sum of the weights must be less than the knapsack's capacity.

Greedy algorithm for knapsack

Algorithm GreedyKnapsack(m,n)
// p[i:n] and [1:n] contain the profits and weights respectively
// if the n-objects ordered such that p[i]/w[i]>=p[i+1]/w[i+1], m size of knapsack and x[1:n] the solution vector
{
For i:=1 to n do x[i]:=0.0
U:=m;
For i:=1 to n do
{
if(w[i]>U) then break;
x[i]:=1.0;
U:=U-w[i];
}
If(i<=n) then x[i]:=U/w[i];
}

Ex: - Consider 3 objects whose profits and weights are defined as(P1, P2, P3) =

(25, 24, 15)

W1, W2, W3) = (18, 15, 10)

n=3 number ofobjects

m=20 Bagcapacity

Consider a knapsack of capacity 20. Determine the optimum strategy for placing the objects in to the knapsack. The problem can be solved by the greedy approach where in the inputs are arranged according to selection process (greedy strategy) andsolve the problem in stages. The various greedy strategies for the problem could be as follows.

(x_1, x_2, x_3)	$\sum x_i w_i$	$\sum x_i p_i$
(1, 2/15, 0)	$18x1+\frac{2}{15}x15 = 20$	$25x1+\frac{2}{15}x24 = 28.2$
(0, 2/3, 1)	$\frac{2}{3}x15+10x1 = 20$	$\frac{2}{3}x24+15x1 = 31$
(0, 1, ½)	$1x15+\frac{1}{2}x10 = 20$	$1x24+\frac{1}{2}x15 = 31.5$
(½, ⅓, ¼)	½ x 18+⅓ x15+ ¼ x10 = 16. 5	½ x 25+⅓ x24+ ¼ x15 = 12.5+8+3.75 = 24.25

Analysis: - If we do not consider the time considered for sorting the inputs then all of the three greedy strategies complexity will be O(n).

UNIT-V: Branch & Bound

Branch and Bound: General method, applications - Travelling sales person problem, 0/1knapsack problem - LC Branch and Bound solution, FIFO Branch and Bound solution.
NP-Hard and NP-Complete problems: Basic concepts, non deterministic algorithms, NP -Hard and NP-Complete classes, Cook's theorem.

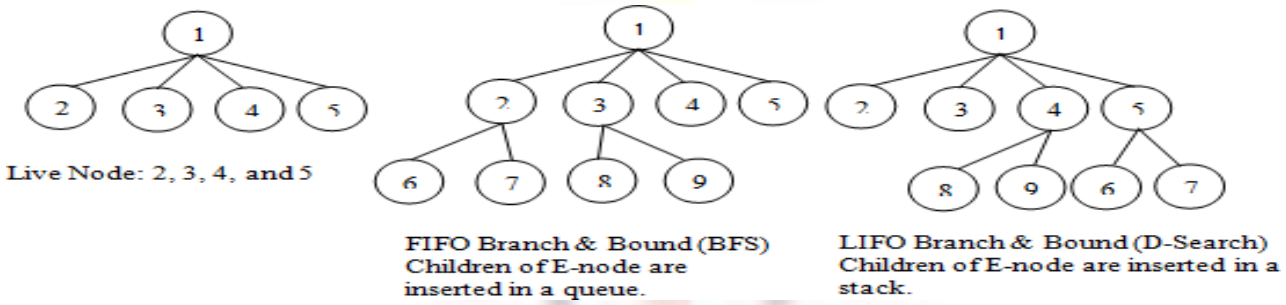
Branch & Bound (B & B) is general algorithm (or Systematic method) for finding optimal solution of various optimization problems, especially in discrete and combinatorialoptimization.

- The B&B strategy is very similar to backtracking in that a state space tree is used to solve a problem.
- The differences are that the B&B method
  - ✓ Does not limit us to any particular way of traversing the tree.
  - ✓ It is used only for optimization problem
  - ✓ It is applicable to a wide variety of discrete combinatorial problem.
- B&B is rather general optimization technique that applies where the greedy method &dynamic programming fail.
- It is much slower, indeed (truly), it often (rapidly) leads to exponential time complexitiesin the worst case.
- The term B&B refers to all state space search methods in which all children of the "E-node" are generated before any other "live node" can become the "E-node"
  - ✓ **Live node→** is a node that has been generated but whose children have not yet

beengenerated.

✓ **E-node**→is a live node whose children are currently being explored.

Dead node→ is a generated node that is not to be expanded or explored any further.Allchildren of a dead node have already been expanded



- ➢ We will use 3-types of search strategies in branch and bound

1) FIFO (First In First Out) search
2) LIFO (Last In First Out) search
3) LC (Least Count) search

FIFO B&B:

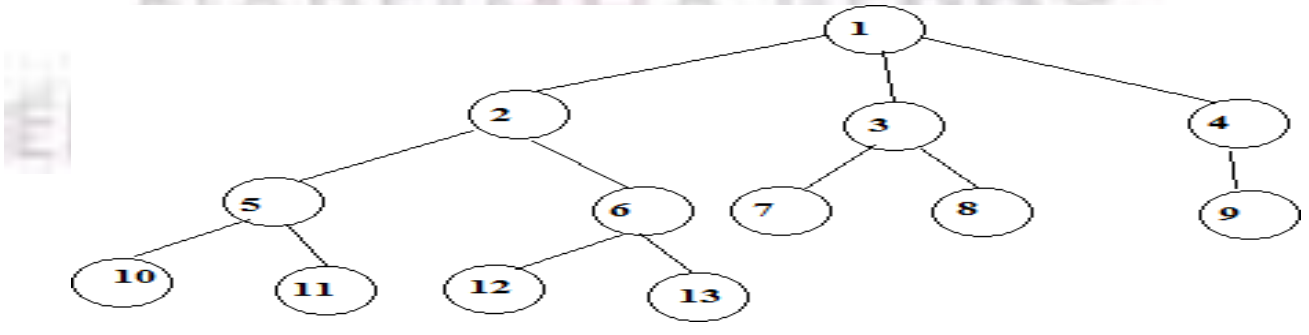
FIFO Branch & Bound is a BFS.

In this, children of E-Node (or Live nodes) are inserted in a queue.

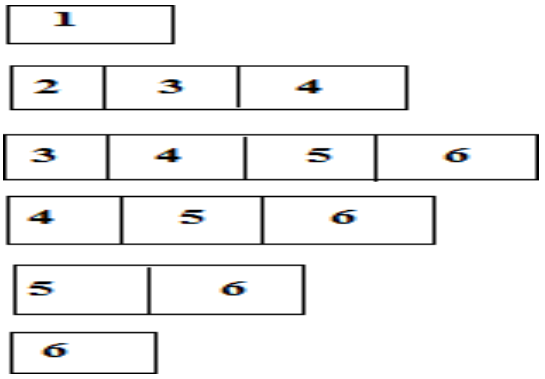
Implementation of list of live nodes as a queue

✓ Least()→ Removes the head of the Queue

✓ Add()→ Adds the node to the end of the Queue



✓ Assume that node '12' is an answer node in FIFO search, 1st we take E-node has '1'

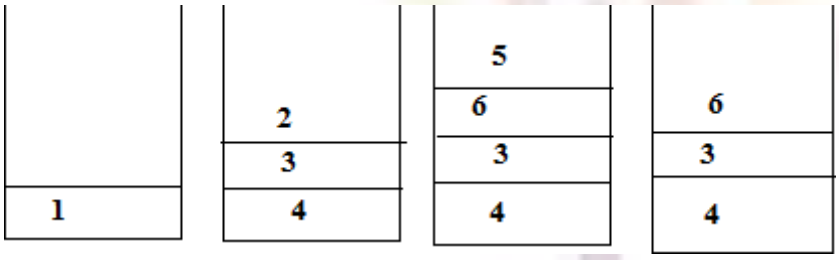


LIFO B&B:

LIFO Brach & Bound is a D-search (or DFS).

In this children of E-node (live nodes) are inserted in a stackImplementation of List of live nodes as a stack

- ✓ Least()→ Removes the top of the stack
- ✓ ADD()→Adds the node to the top of the stack.



Least Cost (LC) Search:

The selection rule for the next E-node in FIFO or LIFO branch and bound is sometimes "blind". i.e., the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.

The search for an answer node can often be speeded by using an "intelligent" ranking function. It is also called an approximate cost function "$\hat{C}$".

Expended node (E-node) is the live node with the best $\hat{C}$ value.

Branching: A set of solutions, which is represented by a node, can be partitioned into mutually (jointly or commonly) exclusive (special) sets. Each subset in the partition is represented by a child of the original node.

Lower bounding: An algorithm is available for calculating a lower bound on the cost of any solution in a given subset.

Each node X in the search tree is associated with a cost: $\hat{C}(X)$

C=cost of reaching the current node, X(E-node) form the root + The cost of reaching an answer node form X.

$$\hat{C}=g(X)+H(X).$$

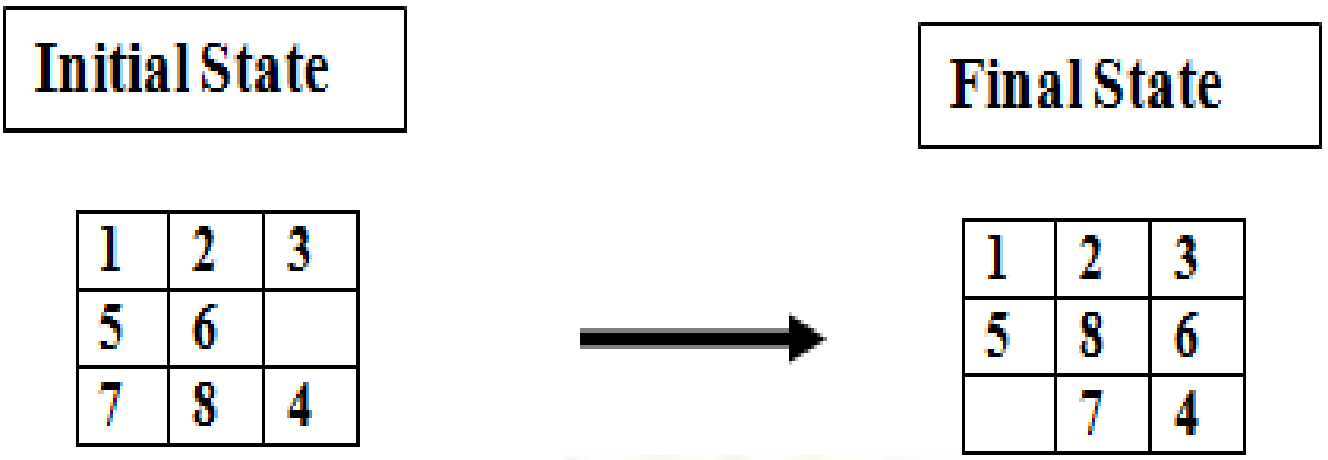
Example:

8-puzzle

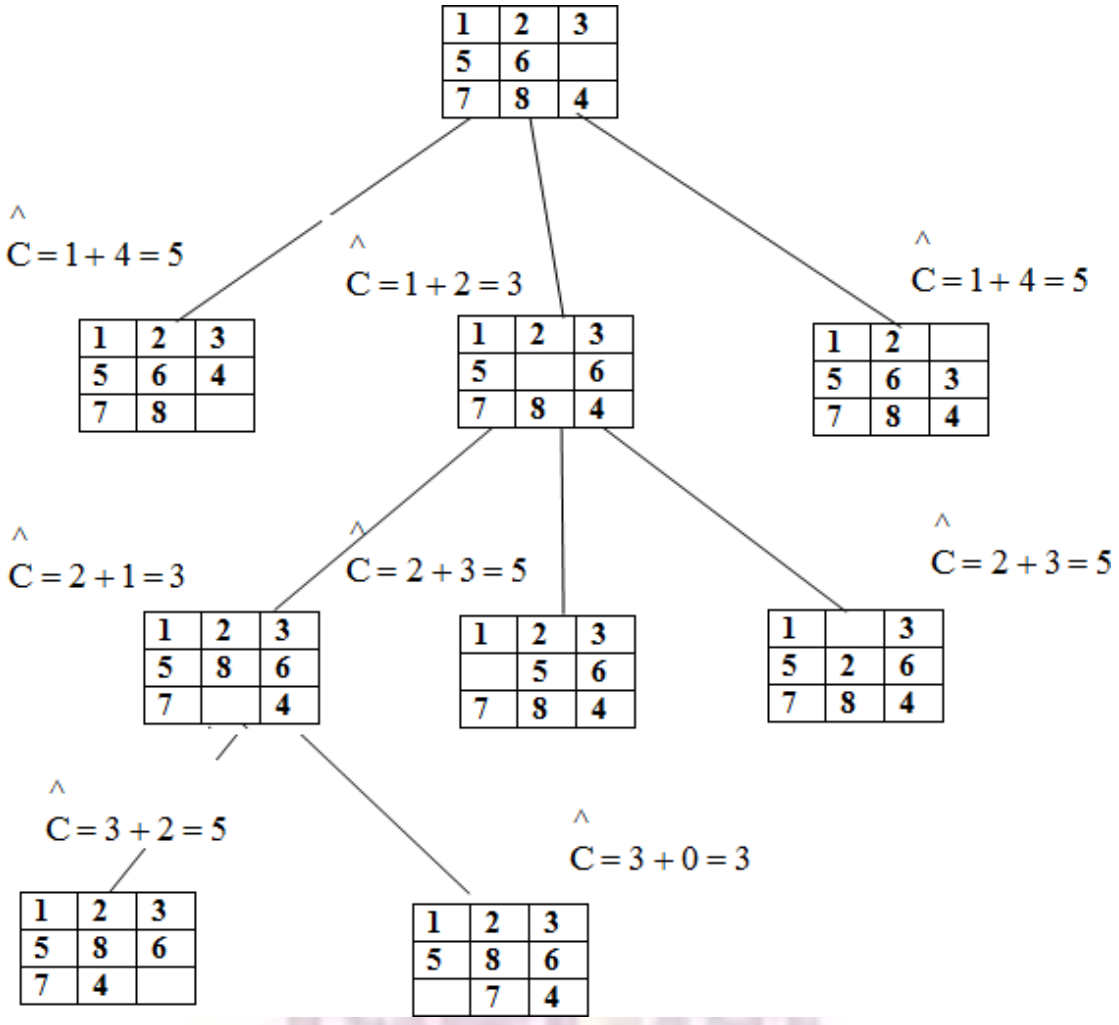
Cost function: $\hat{C} = g(x) + h(x)$

where $h(x)$ = the number of misplaced tiles

and $g(x)$ = the number of moves so farAssumption: move one tile in any



NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE



Travelling Salesman Problem:

Def:- Find a tour of minimum cost starting from a node S going through other nodes only once and returning to the starting point S.

Time Conmlexity of TSP for Dynamic Programming algorithm is $O(n^2 2^n)$

B&B algorithms for this problem, the worest case complexity will not be any better than $O(n^2 2^n)$ but good bunding functions will enables these B&B algorithms to solve some problem instances in much less time than required by the dynamic programming alogrithm.

Let $G=(V,E)$ be a directed graph defining an instances of TSP.Let

C_{ij}→ cost of edge <i, j>

$C_{ij} = \infty$ if <i, j> $\notin$ E

$|V|=n$→ total number of vertices.

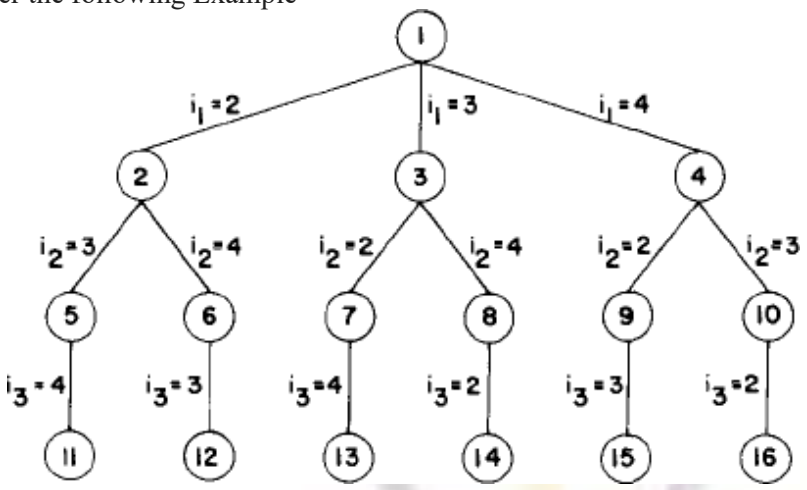
Assume that every tour starts & ends at vertex 1.

Solution Space S= {1, Π , 1 / Π is a permutation of (2, 3. 4 n) } then $|S|=(n-1)!$

The size of S reduced by restricting S

Sothat $(1, i_1, i2, \text{-----} i_{n-1}, 1\} \in S$ iff $<i_j, i_{j+1}> \in E$. $O \le j \le n-1$, $i_0 - i_n = 1$

S can be organized into "State space tree".
Consider the following Example



The above diagram shows tree organization of a complete graph with

|V|=4.
Each leaf node 'L' is a solution node and represents the tour defined by the path from the rootto L.

Node 12 represents the tour.

$i_0=1, i_1=2, i_2=4, i_3=3, i_4=1$

Node 14 represents the tour.

$i_0=1, i_1=3, i_2=4, i_3=2, i_4=1.$

TSP is solved by using LC Branch & Bound:

To use LCBB to search the travelling salesperson "State space tree" first define a costfunction C(.) and other 2 functions $\hat{C}(.)$ & u(.)
Such that $\hat{C}(r) \leq C(r) \leq u(r)$ → for all nodes r.
Cost C(.)→ is the solution node with least C(.) corresponds to a shortest tour in G.

C(A)={Length of tour defined by the path from root to A if A is leaf Cost of a minimum-cost leaf in the sub-tree A, if A is not leaf }

From $\hat{C}(r) \leq C(r)$ then $\hat{C}(r)$ → is the length of the path defined at node A.
From previous example the path defined at node 6 is i_0, i_1, i_2=1, 2, 4 & it consists edge of <1,2> & <2,4>
Abetter $\hat{C}(r)$ can be obtained by using the reduced cost matrix corresponding to G.

- A row (column) is said to be reduced iff it contains at least one zero & remaining entriesare non negative.

A matrix is reduced iff every row & column is reduced.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

(a) Cost Matrix

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

(b) Reduced Cost Matrix

L = 25

Given the following cost matrix:

FIFF 20 30 10 11 1

15 inf 16 4 2

3 5 inf 2 4

I 19 6 18 inf 3 I

L 16 4 7 16 inf

- The TSP starts from node 1: **Node 1**
- Reduced Matrix: To get the lower bound of the path starting at node 1

Row # 1: reduce by 10	Row #2: reduce 2	Row #3: reduce by 2
Finf 10 20 15 inf 16 3 5 inf I19 6 18 i L16 4 7 16	Finf 10 20 13 inf 14 3 5 inf I19 6 18 i L16 4 7 16	Finf 10 20 13 inf 14 1 3 inf I19 6 18 i L16 4 7 16
Row # 4: Reduce by 3:	Row # 5: Reduce by 4	Column 1: Reduce by 1

$\begin{bmatrix} inf & 10 & 20 \\ 13 & inf & 14 \\ 1 & 3 & inf \\ 16 & 3 & 15 \\ 16 & 4 & 7 \end{bmatrix}$	$\begin{bmatrix} inf & 10 & 20 \\ 13 & inf & 14 \\ 1 & 3 & inf \\ 16 & 3 & 15 \\ 12 & 0 & 3 \end{bmatrix}$	$\begin{bmatrix} inf & 10 & 20 \\ 12 & inf & 14 \\ 0 & 3 & inf \\ 15 & 3 & 15 \\ 11 & 0 & 3 \end{bmatrix}$
Column 2: It is reduced.	Column 3: Reduce by 3 $\begin{bmatrix} inf & 10 & 17 & 0 & 1 \\ 12 & inf & 11 & 2 & 0 \\ 0 & 3 & inf & 0 & 2 \\ 15 & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & 12 & inf \end{bmatrix}$	Column 4: It is reduced. Column 5: It is reduced.

The reduced cost is: RCL = 25
So the cost of node 1 is: Cost (1) = 25
The reduced matrix is:

Cost (1) = 25

$$\begin{bmatrix} inf & 10 & 17 & 0 & 1 \\ 12 & inf & 11 & 2 & 0 \\ 0 & 3 & inf & 0 & 2 \\ 15 & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & 12 & inf \end{bmatrix}$$

- ***Choose to go to vertex 2: Node 2***

  - Cost of edge <1,2> is: A(1,2) = 10
  - Set row #1 = inf since we are choosing edge <1,2>
  - Set column # 2 = inf since we are choosing edge <1,2>
  - Set A(2,1) = inf
  - The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & 1 \\ inf & inf & 11 & 2 & 0 \\ 15 & inf & inf & 0 & 0 \\ 11 & inf & 12 & & \end{bmatrix}$$

  - The matrix is reduced:
  - RCL = 0
  - The cost of node 2 (Considering vertex 2 from vertex 1) is:

Cost(2) = cost(1) + A(1,2) = 25 + 10 = 35

- Cost of edge <1,3> is: A(1,3) = 17 (In the reduced matrix
- Set row #1 = inf since we are starting from node 1
- Set column # 3 = inf since we are choosing edge <1,3>
- Set A(3,1) = inf
- The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf & 1 \\ inf & 3 & inf & 2 & 0 & \\ 15 & 3 & inf & 0 & 0 & \\ inf & 12 & & & & \end{bmatrix}$$

Reduce the matrix: Rows are reduced

The columns are reduced except for column #

The lower bound is: RCL = 11

The cost of going through node 3 is:

Choose to go to vertex 3: Node 3:

Choose to go to vertex 4: **Node 4**

Remember that the cost matrix is the one that was reduced at the starting vertex 1Cost of edge <1,4> is: A(1,4) =0

Set row #1 = inf since we are starting from node 1

Set column # 4 = inf since we are choosing edge <1,4>Set A(4,1) = infThe

resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf & 1 \\ 12 & inf & 11 & inf & 0 & \\ 0 & 3 & inf & inf & 2 & \\ inf & 3 & 12 & inf & 0 & \\ 11 & 0 & 0 & inf & inf & \end{bmatrix}$$

Reduce the matrix: Rows are reduced

The lower bound is: RCL = 0
The cost of going through node 4 is:
cost(4) = cost(1) + RCL + A(1,4) = 25 + 0 + 0 = 25

➢ ***Choose to go to vertex 5: Node 5***

- Remember that the cost matrix is the one that was reduced at starting vertex 1
- Cost of edge <1,5> is: A(1,5) = 1
- Set row #1 = inf since we are starting from node 1
- Set column # 5 = inf since we are choosing edge <1,5>
- Set A(5,1) = inf
- The resulting cost matrix is:

$$\begin{matrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & 2 & inf \end{matrix}$$

$$\begin{matrix} 0 & 3 & inf & 0 & inf \\ 15 & 3 & 12 & inf & \\ inf & 0 & 0 & 12 & \end{matrix}$$

Reduce the matrix:

Reduce rows:

Reduce row #2: Reduce by 2

$$\begin{matrix} inf & inf & inf & inf & inf \\ 10 & inf & 9 & 0 & inf \end{matrix}$$

$$\begin{matrix} 0 & 3 & inf & 0 & \\ 15 & 3 & 12 & inf & \\ inf & 0 & 0 & 12 & \end{matrix}$$

Reduce row #4: Reduce by 3

$$\begin{matrix} & & & & \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & \infty & 0 & 12 & \infty \end{matrix}$$

Columns are reduced
The lower bound is: RCL = 2 + 3 = 5
The cost of going through node 5 is:
cost(5) = cost(1) + RCL + A(1,5) = 25 + 5 + 1 = 31

In summary:
So the live nodes we have so far are:

- ✓ 2: cost(2) = 35, path: 1->2
- ✓ 3: cost(3) = 53, path: 1->3
- ✓ 4: cost(4) = 25, path: 1->4
- ✓ 5: cost(5) = 31, path: 1->5

Explore the node with the lowest cost: Node 4 has a cost of 25Vertices to be explored from node 4: 2, 3, and 5
Now we are starting from the cost matrix at node 4 is:

Cost (4) = 25

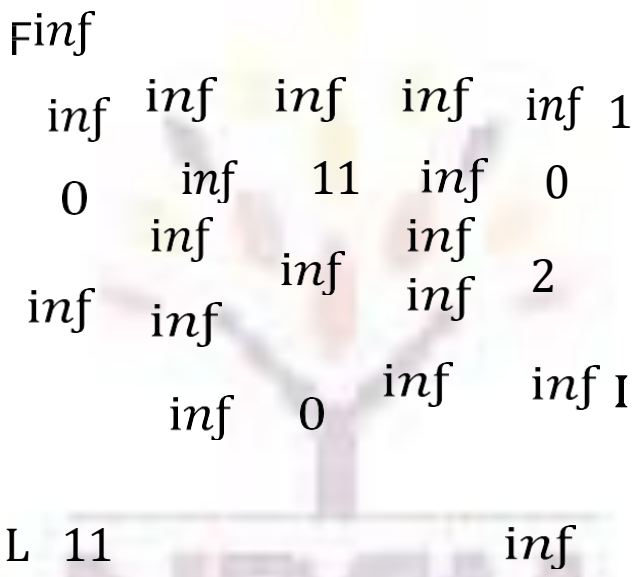
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

- **Choose to go to vertex 2: Node 6 (path is 1->4->2)**

Cost of edge <4,2> is: A(4,2) = 3

Set row #4 = inf since we are considering edge <4,2>

Set column # 2 = inf since we are considering edge <4,2>
Set A(2,1) = inf



Reduce the matrix: Rows are reduced

Columns are
reducedThe lower bound is: RCL = 0

The cost of going through node 2 is:

- **Choose to go to vertex 3: Node 7 (path is 1->4->3)**

Cost of edge <4,3> is: A(4,3) = 12

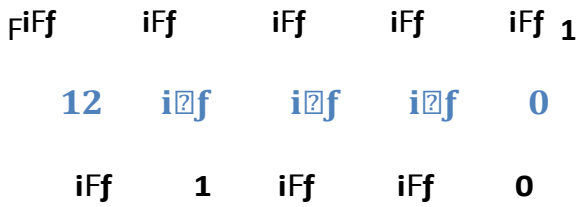
Set row #4 = inf since we are considering edge <4,3>

Set column # 3 = inf since we are considering edge <4,3>

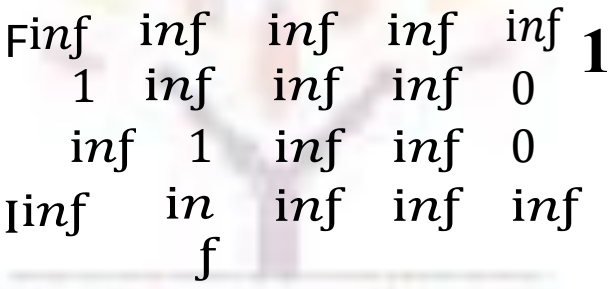
$$\begin{matrix} & & & & \\ 12 & inf & inf & inf & 0 \\ & & & & \\ inf & inf & inf & inf & inf \\ & & & & \end{matrix}$$

NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE

Reduce row #3: by 2:



Reduce column # 1: by 11



The lower bound is: RCL = 13
So the RCL of node 7 (Considering vertex 3 from vertex 4) is:Cost(7) = cost(4) + RCL + A(4,3) = 25 + 13 + 12 = 50

NRCM
NARSIMHA REDDY ENGINEERING COLLEGE

- Choose to go to vertex 5: **Node 8** (path is 1->4->5)

Cost of edge <4,5> is: A(4,5) = 0

Set row #4 = inf since we are considering edge <4,5>

Set column # 5 = inf since we are considering edge <4,5>

$$\begin{matrix} inf & inf & inf & inf & 1 \\ & inf & 11 & inf & inf \\ F inf & & & & \\ & 3 & inf & inf & inf \\ L & inf & & & \end{matrix}$$

Reduce the matrix:

Reduced row 2: by 11

$$\begin{matrix} & inf & inf & inf & 1 \\ & 0 & inf & inf & \\ F inf & inf & & & \\ & inf & inf & inf & \\ 1 & inf & 0 & inf & inf \\ 0 & 3 & & & \\ L & inf & 0 & & \end{matrix}$$

In summary: So the live nodes we have so far are:

- ✓ 2: cost(2) = 35, path: 1->2
- ✓ 3: cost(3) = 53, path: 1->3
- ✓ 5: cost(5) = 31, path: 1->5
- ✓ 6: cost(6) = 28, path: 1->4->2
- ✓ 7: cost(7) = 50, path: 1->4->3
- ✓ 8: cost(8) = 36, path: 1->4->5
- Explore the node with the lowest cost: Node 6 has a cost of 28
- Vertices to be explored from node 6: 3 and 5
- Now we are starting from the cost matrix at node 6 is:

Cost (6) = 28

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & 11 & inf & 0 \\ 0 & inf & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & inf & 0 & inf & inf \end{bmatrix}$$

NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE

➢ ***Choose to go to vertex 3: Node 9 (path is 1->4->2->3)***

Cost of edge <2,3> is: A(2,3) = 11

Set row #2 = inf since we are considering edge <2,3>

Set column # 3 = inf since we are considering edge <2,3>Set A(3,1) = inf

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & inf & inf & inf & inf \end{bmatrix} 1$$

The resulting cost matrix is:

Reduce the matrix: Reduce row #3: by 2

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 11 & inf & inf & inf & inf \end{bmatrix}$$

Reduce column # 1: by 11

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \end{bmatrix} 1$$

I

1

NARSIMHA REDDY ENGINEERING COLLEGE

ɪi□ƒ i□ƒ i□ƒ i□ƒ i□ƒ ɪL
0 i□ƒ i□ƒ i□ƒ i□ƒ

The lower bound is: RCL = 2 +11 = 13

So the cost of node 9 (Considering vertex 3 from vertex 2) is:Cost(9) = cost(6) + RCL + A(2,3) = 28 + 13 + 11 = 52

NRCM
NARSIMHA REDDY
ENGINEERING COLLEGE

- ***Choose to go to vertex 5: Node 10 (path is 1->4->2->5)***

Cost of edge <2,5> is: A(2,5) = 0

Set row #2 = inf since we are considering edge <2,3>

Set column # 3 = inf since we are considering edge <2,3>
Set A(5,1) = inf

$$\begin{matrix} Finf & inf & inf & inf & 1\ inf \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \\ inf & inf & inf & inf & Inf \\ inf & inf & inf & inf & Inf \end{matrix}$$

Reduce the matrix: Rows reduced

Columns reduced

The lower bound is: RCL = 0

So the cost of node 10 (Considering vertex 5 from vertex 2) is:
Cost(10) = cost(6) + RCL + A(2,3) = 28 + 0 + 0 = 28

In summary: **So the live nodes we have so far are:**

- ✓ 2: cost(2) = 35, path: 1->2
- ✓ 3: cost(3) = 53, path: 1->3
- ✓ 5: cost(5) = 31, path: 1->5
- ✓ 7: cost(7) = 50, path: 1->4->3
- ✓ 8: cost(8) = 36, path: 1->4->5
- ✓ 9: cost(9) = 52, path: 1->4->2->3

- ✓ 10: cost(2) = 28, path: 1->4->2->5
- ➢ Explore the node with the lowest cost: Node 10 has a cost of 28
- ➢ Vertices to be explored from node 10: 3
- ➢ Now we are starting from the cost matrix at node 10 is:

Cost (10)=28

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & 0 & inf & inf \end{bmatrix}$$

- ➢ **<u>Choose to go to vertex 3: Node 11 (</u> path is 1->4->2->5->3)**

Cost of edge <5,3> is: A(5,3) = 0

Set row #5 = inf since we are considering edge <5,3>

Set column # 3 = inf since we are considering edge <5,3>

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \end{bmatrix}$$

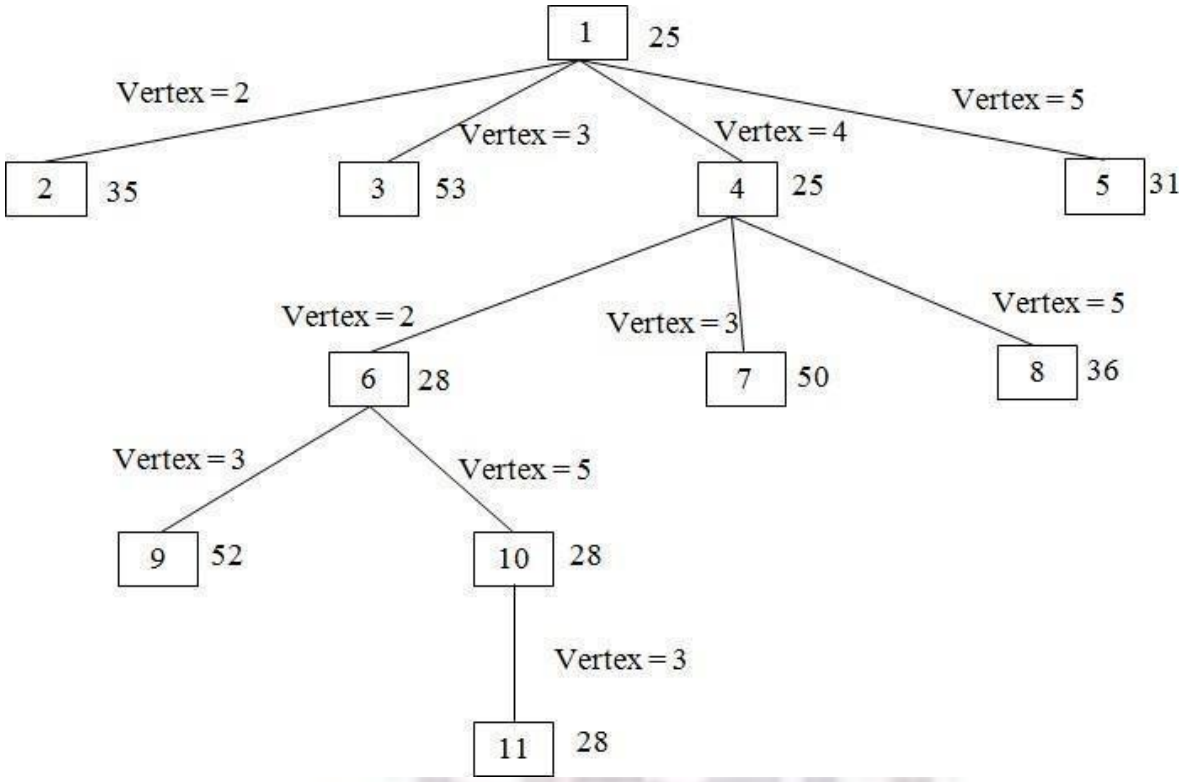
Reduce the matrix: Rows reduced

Columns reduced

The lower bound is: RCL = 0

So the cost of node 11 (Considering vertex 5 from vertex 3) is:

State Space Tree:



0/1 Knapsack Problem

What is Knapsack Problem: Knapsack problem is a problem in combinatorial optimization,Given a set of items, each with a mass & a value, determine the number of each item to include in a collection so thatthe total weight is less than or equal to a given limit & the total value is aslarge as possible.

O-1 Knapsack Problem can formulate as. Let there be n items, Z_1 **to** Z_n where Z_i has value

P_i & weight w_i. The maximum weight that can carry in the bag is m.All values and weights arenon negative.
Maximize the sum of the values of the items in the knapsack, so that sumof the weights mustbe less than the knapsack's capacity m.
The formula can be stated as

$$\text{maximize} \sum_{1 \le i \le n} p_i x_i$$

$$\text{subject to} \sum_{1 \le i \le n} w_i x_i \le M$$

To solve o/1 knapsack problem using B&B:

- Knapsack is a maximization problem
- Replace the objective function $\sum p_i x_i$ bythe function $-\sum p_i x_i$ to make it into aminimiza
- The modified knapsack problem is stated as

$$\text{Minimize } -\sum_{i=1}^{n} p_i x_i$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i < m,$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

- Fixed tuple size solution space:
  - Every leaf node in state space tree represents an answer for which $\sum_{1 \le i \le n} w_i x_i \leqslant m$ is an answer node; other leaf nodes are infeasible
  - For optimal solution, define

$$c(x) = -\sum_{1 \le i \le n} p_i x_i$$

for every answer node x

- For infeasible leaf nodes, $c(x) = \infty$
- For non leaf nodes

  c(x) = min{c(lchild(x)), c(rchild(x))}

- Define two functions cˆ(x) and u(x)

such that for everynode x,

cˆ(x) ≤ c(x) ≤ u(x)

- Computing cˆ(·) and u(·)

Let x be a node at level j, $1 \le j \le n+1$

Cost of assignment: $-\sum_{1 \le i < j} p_i x_i$

$c(x) \le -\sum_{1 \le i < j} p_i x_i$

We can use $u(x) = -\sum_{1 \le i < j} p_i x_i$

Using $q = -\sum_{1 \le i < j} p_i x_i$, an improved upper bound function $u(x)$ is

$$u(x) = \text{ubound}(q, \sum_{1 \le i < j} w_i x_i, j-1, m)$$

```
Algorithm ubound ( cp, cw, k, m )

{

// Input: cp: Current profit total

// Input: cw: Current weight total

// Input: k: Index of last removed item

// Input: m: Knapsack capacity

b=cp; c=cw;
for i:=k+1
    to n
    do{
    if(c+
    w[i] ≤
    m)
    then {
            c:=c+w[i]; b=b-p[i];
    }
}
return b;
}
```

NP-Hard and NP-Complete problems: Basic concepts, non deterministic algorithms, NP -Hard and NPComplete classes, Cook'stheorem.

Basic concepts:

NP, Nondeterministic Polynomial time

The problems has best algorithms for their solutions have "Computing times", that clusterinto two groups

Group 1	Group 2

> Problems withsolution time bound by a polynomial of a small degree. > It also called "Tractable Algorithms" > Most Searching& Sortingalgorithms are polynomial time algorithms > **Ex:** Ordered Search **(O (log n)),** Polynomial evaluation **O(n)** Sorting **O(n.log n)**	> Problems with solution times not bound by polynomial (simply non polynomial) > Thesearehardorintractable problems > None of the problems in this group has been solved by any polynomial time algorithm > **Ex:** Traveling Sales Person **$O(n^2 2^n)$** Knapsack **$O(2^{n/2})$**

o one has been ableto develop a polynomial time algorithm for anyproblem in the 2nd group (i.e., group 2)

So, it is compulsory and finding algorithms whose computing times aregreater than polynomial very quickly because such vast amounts of timeto execute that even moderatesize problems cannot be solved.

Theoryof NP-Completeness:

Showthat may of the problems with no polynomial timealgorithms are computational timealgorithms are computationally related.

There are two classes of non-polynomial time problems

1. NP-Hard
2. NP-Complete
3. **NP Complete Problem:** A problem that is NP-Complete can solvedin polynomial time ifand only if (iff) all other NP-Complete problems can also be solved in polynomial time.
4. **NP-Hard:** Problem can be solved in polynomial time then all NP-Complete problems can besolved in polynomial time.
5. All NP-Complete problemsare NP-Hard but some NP-Hard problemsarenot know to be NP-Complete.

Nondeterministic Algorithms:

Algorithms with the property that the result of every operation is uniquelydefined are termedas deterministic algorithms. Such algorithms agree withthe way programs are executed on a computer.

Algorithms which contain operations whose outcomes are not uniquely defined but arelimited to specified set of possibilities. Such algorithmsare called nondeterministic algorithms.

The machine executing such operations is allowed to choose any one of these outcomessubject to a termination condition to be definedlater.

To specify nondeterministic algorithms,

thereare 3 new functions. Choice(S) ,

arbitrarily chooses one of the

elements of sets S Failure (), Signals

an Unsuccessful completion

Success (), Signals a successful completion.

Example for Non Deterministic algorithms:

Algorithm Search(x){
//Problem is to search an element x
//output J, suchthat A[J]=x; or J=0 if x is not in A
J:=Choice(1,n);
if(A[J]:=x) then {
Write(J);
Success();
}
else{
write(0);
failure();
}
	Whenever there is a set of choices that leads to a successful completion then one such set of choices is always made and the algorithm terminates.
	A Nondeterministic algorithm terminates unsuccessfully if and only if (iff) there exists no set of choices leading to a successful signal.

Nondeterministic Knapsack algorithm	

Algorithm DKP(p, w, n, m, r, x){
W:=0;
P:=0;
for i:=1 to n do{
x[i]:=choice(0, 1);
W:=W+x[i]*w[i];
P:=P+x[i]*p[i];
}
if((W>m) or (P<r)) then Failure();
else Success();
}
	p) given Profits
	w) given Weights
	n) Number of elements (number of p or w)
	m) Weight of bag limit
	P) Final Profit
	W) Final weight

The Classes NP-Hard & NP-Complete:

For measuring the complexity of an algorithm, we use the input length asthe parameter. Forexample, An algorithm A is of polynomial complexityp() such that the computing time of Ais O(p(n)) for every input of size n.**Decision problem/ Decision algorithm:** Any problem for which the answer is either zero orone is decision problem. Any algorithm for a decision problem is termed a decision algorithm.

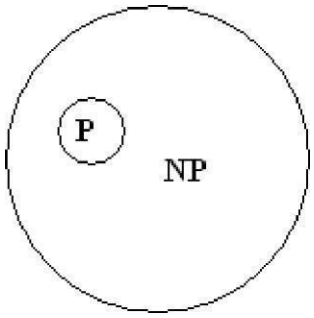
Optimization problem/ Optimization algorithm: Any problem thatinvolves the identification of an optimal (either minimum or maximum)

value of a given cost function isknown as an optimization problem.An optimization algorithm is used to solve an optimization problem.

P) is the set of all decision problems solvable by deterministic algorithms in polynomial time.

NP) is the set of all decision problems solvable by nondeterministic algorithms in polynomial time.

Sincedeterministicalgorithms are just a special case of nondeterministic, by this weconcluded that **P** ⊂ **NP**



Commonly believed relationship between P & NP

The most famous unsolvable problems in Computer Science is Whether P=NP or P≠NPIn considering this problem, s.cook formulated the following question.

If there any single problem in NP, such that if we showed it to be in 'P' then that wouldimply that P=NP.

Cook answered this question with

Theorem: Satisfiability is in P if and only if (iff) P=NP

-)Notation of Reducibility

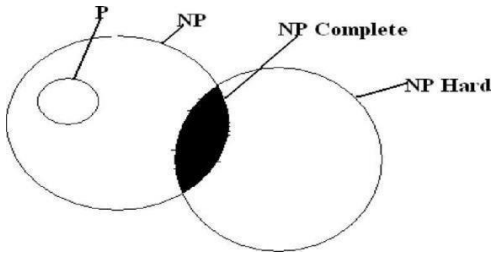
Let L_1 and L_2 be problems, Problem L_1 reduces to L_2 (written $\mathbf{L_1 \alpha L_2}$) iff there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm thatsolves L_2 in polynomialtime

This implies that, if we have a polynomial time algorithm for L_2, Thenwe can solve L_1 inpolynomial time.

Here α-) is a transitive relation i.e., $\mathbf{L_1 \alpha L_2}$ and $\mathbf{L_2 \alpha L_3}$ then $\mathbf{L_1 \alpha L_3}$

A problem L is NP-Hard if and only if (iff) satisfiability reduces to L ie., **Statisfiability α L**

A problem L is NP-Complete if and only if (iff) L is NP-Hard and **L Є NP**



Commonly believed relationship among P, NP, NP-Completeand NP-

HardMost natural problems in NP are either in P or NP- complete.

Examples of NP-complete problems:

- Packing problems: SET-PACKING, INDEPENDENT-SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3-COLOR, CLIQUE.
- Constraint satisfaction problems: SAT, 3-SAT.

Numerical problems: SUBSET-SUM, PARTITION, KNAPSACK

Cook's Theorem: States that satisfiability is in P if and only if P=NP

IfP=NPthen satisfiability is in P

If satisfiability is in P, then P=NPTo do this

- A-) Any polynomial time nondeterministic decision algorithm.

  I-)Input of that algorithm

  Then formula Q(A, I), Such that Q is satisfiable iff '**A'** has a successfultermination with Input **I**.

- If the length of '**I**' is '**n'** and the time complexity of **A** is **p(n)** for some polynomial **p()** then length of Q is $\mathbf{O(p^3(n) \log n)=O(p^4(n))}$

  The time needed to construct Q is also $\mathbf{O(p^3(n) \log n).}$

- A deterministic algorithm '**Z'** to determine the outcome of '**A**' on any input '**I**'Algorithm **Z** computes '**Q**' and then uses a deterministic algorithm for the satisfiability problem to determine whether '**Q**' is satisfiable.If **O(q(m))** is the time needed to determine whether a formula of length '**m**' issatisfiable then the complexity of '**Z**' is $\mathbf{O(p^3(n) \log n + q(p^3(n) \log n)).}$

  - If satisfiability is '**p**', then '**q(m)**' is a polynomial function of '**m**' and thecomplexity of '**Z**' becomes '**O(r(n))'** for some polynomial '**r()'.**

- Hence, if satisfiability is in **p**, then for every nondeterministic algorithm **A** in **NP**, wecan obtain a deterministic **Z** in **p.**

By this we shows that satisfiability is in **p** then **P=NP**